A STEWART PLATFORM SIX-AXIS MILLING MACHINE DEVELOPMENT

A thesis

submitted in fulfilment

of the requirements for the degree

of

Master of Engineering

in the University of Canterbury

by

Geoffrey P. Rathbun

Supervisors: Professor H. McCallion

: Dr G. R. Dunlop

University of Canterbury

1986

This work is dedicated to my parents for their ever-present support.

CONTENTS

Contents (continued)

Contents (continued)

ABSTRACT

This manuscript describes the development of an experimental NC milling machine having control over all the 6 bodily degrees of freedom of movement.

The mechanism used is called a Stewart Platform. These mechanisms have a platform connected to a base by 6 variable length "legs". The Stewart Platform is considered to have good potential as a rigid 5 axis milling machine having a simpler structure than conventional machines.

Platform motion is achieved by pre-calculated movements of 6 stepper motors which actuate leg mounted lead screws. These movements are controlled using microcomputer controlled electronic hardware.

The development of the system progressed along several paths among which were: path synthesis methods in six dimensions, simultaneous open-loop control of 6 stepper motors, data manipulation and storage structures in software to execute the control strategies, and the development of the existing positioning machine for milling operations.

The system was successfully developed to enable machining of rigid urethane foam blocks in 5 useful degrees of freedom. The envelope of movement was restricted to +/- 100mm translations and +/- 30 degree rotations. Accuracy of movement was beyond that detectable using foam.

Suggestions are made for further developments, particularly for improving the machine's envelope of movement and method of actuation.

SYMBOLS

		Units
	CHAPTER 3	
$\underline{AS}$	Actuator State Vector (6 x 1)	...steps
$\underline{C}$	General coordinate in SS (6 x 1)	...mm & radians
$\underline{C}_S$	Starting coordinate (6 x 1)	...mm & radians
$\underline{C}_F$	Finishing coordinate (6 x 1)	...mm & radians
e	Movement error off a straight line	... mm
INT	Function taking closest lower integer	
[I]	Identity Matrix (3 x 3)	
k	Number of part of a path	
$\underline{l}$	Leg length vector (6 x 1)	...mm
M	Linear movement scalar	...mm
m	Allowable path vector length	...mm
N	Number of parts to a path	
N_i	Step state of actuator i	...steps
$\underline{P}$	Position vector in three dimensional reference coordinate system	...mm
$\underline{p}$	Position vector fixed to moveable coordinate system	...mm
[R]	Orientation operator matrix (3 x 3)	
SS	Six space; spans all positional and orientational states of a body	
T	Time allowed for a movement	...seconds

Symbols (continued)

		Units
$\underline{T}$	Position vector of moveable coordinate system origin	...mm
u	Reference system X coordinate	...mm
v	Reference system Y coordinate	...mm
w	Reference system Z coordinate	...mm
ϕ	First Euler angle	...radians
θ	Second Euler angle	...radians
θ_D	Stepper motor unit step angle	...radians
θ_H	Hooke joint error angle	...mm & radians
ψ	Third Euler angle	...mm & radians

CHAPTER 4

ACU	Actuator Control Unit	
AS	as Chapter 3	
CD	Common Divisor theory	
CCG	Counter Comparator Group	
CCS	Computer Control System	
CPATE	Controlled Pulse Arrival Time Error Theory	
D_i	Direction signal on line i	...logic
D1...D6	Direction lines	
DIRG	8-bit direction and gating word	...logic
e_i	Arrival time error of motor i	...seconds
F	Oscillator frequency	...Hz
k	Pulse arrival error	...pulse
$\underline{M}$	Movement vector (6 x 1)	...steps
MCU	Machine Control Unit	

Symbols (continued)

		Units
N	as Chapter 3	
N_i	Steps output on actuator i	...steps
OSFDG	Overall Speed Frequency Divider Group	
P_i	Position signal on line i	...pulse
P1...P6	Position lines	
RSD_i	Relative speed divisor for actuator i	...integer
RSFDG	Relative Speed Frequency Diivider Group	
SDIV	Speed Divisor	...integer
T	as Chapter 3	
t	Present time	...seconds
TARGET	Target step count value	...integer
V_i	Pulse speed on position line i	...pulses/second
	Stepper motor shaft position	...radians

CHAPTER 5

AS	As Chapters 3 and 4	
ASP	Present actuator state array (6 integers)	...steps
ASZ	Actuator zero state array (6 integers)	...steps
ASSC	Commanded actuator starting state array (6 integers)	...steps
C_S, C_F	as Chapter 3	
CCG	as Chapter 4	

Symbols (continued)

		Units
CCS	as Chapter 4	
GPIC	Advanced Micro Devices "General Purpose Interrupt Controller Chip"	
k	as Chapter 3	
l	Euler matrix parameter	
m	Euler matrix parameter	
MIU	Movement Information Unit	
MCDB	Machine Control Data Block	
N	as Chapters 3 and 4	
n	Euler matrix parameter	
OSFDG	as Chapter 4	
$\underline{p}$	as Chapter 3	
R	Path resolution	...mm
$[R]_k$	Orientation matrix operator for the k'th position of $\underline{p}$	
RSFDG	as Chapter 4	
STC	System Timing Controller chip (A.M.D.)	
$\underline{T}$	as Chapter 3	
V	Path speed	...mm/second
VDU	Visual Display Unit	
α	Euler matrix rotation parameter	...radians

Symbols (continued)

		Units
	CHAPTER 6	
A_i	Acceleration of motor i across a system node (in ratio terms)	
BA	Number of system vectors over which first acceleration event takes place	
BB	Number of systems vectors over which second acceleration event takes place	
K_a	Acceleration constant	
K_d	Deceleration constant	
LASTM(i)	Number of steps output on motor i in the last system vector	...steps
LASTM(LSTFST)	as above, but last fastest motor	...steps
M(i)	Number of steps output on motor i in the present system vector	...steps
N	Number of system vectors in a path vector	
V_i	General speed of motor i	
VE	Maximum allowable speed of fastest motor at end of a path vector	...steps/sec
VEA	Maximum allowable speed of fastest motor for a trans-node acceleration	...steps/sec

Symbols (continued)

		Units
VED	as above but for deceleration	...steps/sec
VEDC	as above but for a direction change	...steps/sec
VT	Target speed of fastest motor	...steps/sec
VT_{max}	Maximum allowable target speed	...steps/sec

CHAPTER 1

Introduction

The subject for this report was first introduced to the author as a requirement to achieve full motion control over a six degree of freedom positioning machine residing in the Applied Mechanics Laboratory of the Mechanical Engineering Department of the University of Canterbury.

The machine had been constructed originally in 1978 for the purpose of manipulating objects for an automatic assembly research project. (see reference [1]) The machine is shown pictorially in Figure 1.1 and has a range of movement of about +/- 100mm in the linear axes and about +/- 30 degrees in the rotational axes. This type of machine, having a platform supported on six variable length legs is called a Stewart Machine. It first appeared in England in the 1950's. (see reference [2])

Proper control over the machine was not originally achieved due to a comparatively cumbersome computer control system and errors in the motion synthesis algorithms.

The author's major objective was to achieve full control of the machine using a Z80 CPU based microcomputer system plus appropriate hardware.

It was considered judicious to broaden the scope of the work outside the above objective. To this end it was envisaged that the six degree of freedom (6 D.O.F.) positioning machine would be used in a system that required or would benefit from the machine's special

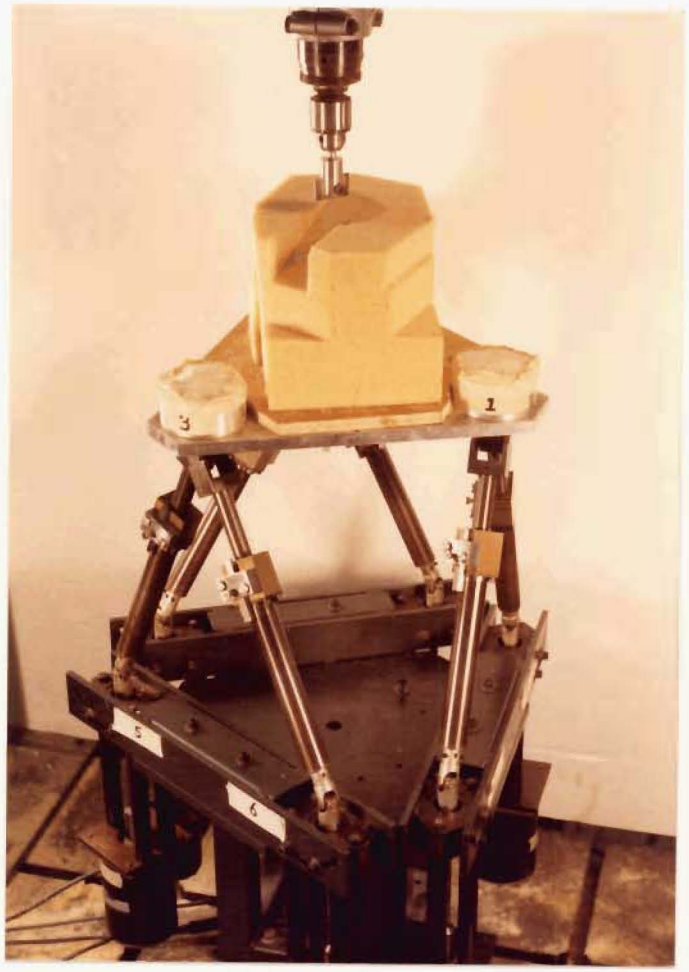


FIGURE 1.1

The University of Canterbury Stewart Platform

attributes. Development of the control system would then hopefully benefit from appropriate technology and concepts. (See [3] and [4] for examples)

The special attributes of the Stewart machine may be summarised as:

(1) Simultaneous control of all the 6 D.O.F. of the platform is possible,

(2) The machine structure yields a very stiff connection between the platform and the base.

Special attributes of our particular configuration are:

(a) The use of high resolution stepper motors to vary the machine geometry yields a highly accurate positioning capability (to about 0.01mm),

(b) The stepper motors allow a moderate speed of movement, (up to 15 mm/sec),

(c) An open loop digital control system can be used with the stepper motors, without sacrificing accuracy or repeatability.

Chapter 2 discusses the above points in relation to the present uses of the Stewart machine and to other possible uses, and concludes that the machine should be developed with a view to creating a milling machine with six degrees of freedom. (Though only five are usable)

The following chapters deal with, in turn, the development of a general path synthesis method (Chapter 3), the development of the machine control system (Chapters 4 to 6), the operation of the system (Chapter 7) and finally a "future system" proposal (Chapter 8).

To aid the understanding of the following chapters, it is suggested the reader peruse Chapter 7 describing the operation of the completed system shown in Figure 1.2.

FIGURE 1.2

Views of the UoC Stewart Platform Milling System

CHAPTER 2

SIX DEGREE OF FREEDOM MACHINES AND THEIR USES

2.1 Comparison of Existing Machines

The existing machines which have control of all six degrees of movement freedom are:

(1) Open Kinematic Chain Robots: These have various forms of variable geometry. They have, in the simplest cases, 6 fixed length links each joined by a revolute joint having 1 degree of freedom. Such a robot is shown in Figure 2.1.

(2) Milling Machines: These usually have a maximum of 5 degrees of freedom, the maximum requirement for milling operations, but are included here because a comparison to Groups 1 and 3 can be made by assuming the addition of a rotary table to the milling machine to allow it to achieve full control over all six D.O.F.. A typical machine is shown in Figure 2.2.

These machines typically have three or more prismatic linearly variable joints for control of the linear degrees of freedom, and 3 rotary joints for control of the three rotational degrees of freedom. The three linear and three rotary joints with interposing links are arranged in a variety of ways, often as two serially linked chains, each having complementary D.O.F.. In these cases, the six D.O.F. are not with respect to a fixed reference point, but to the

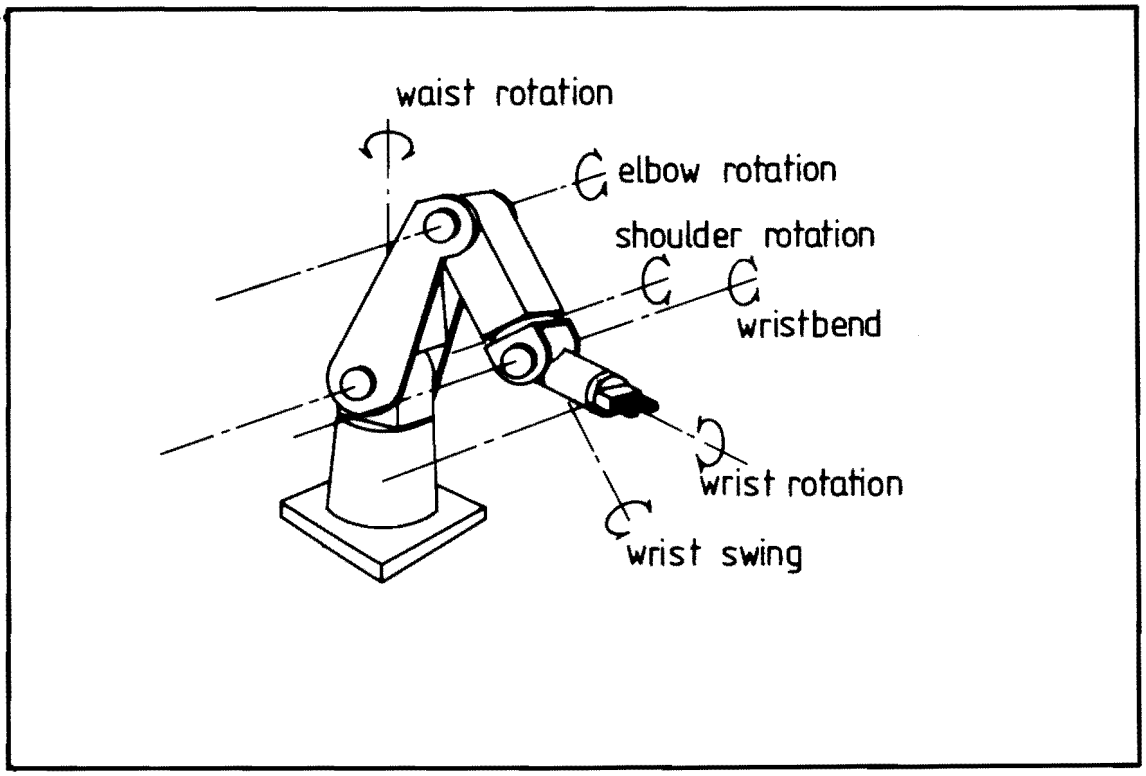


Figure 2.1 A Serial Link Open Chain Robot.

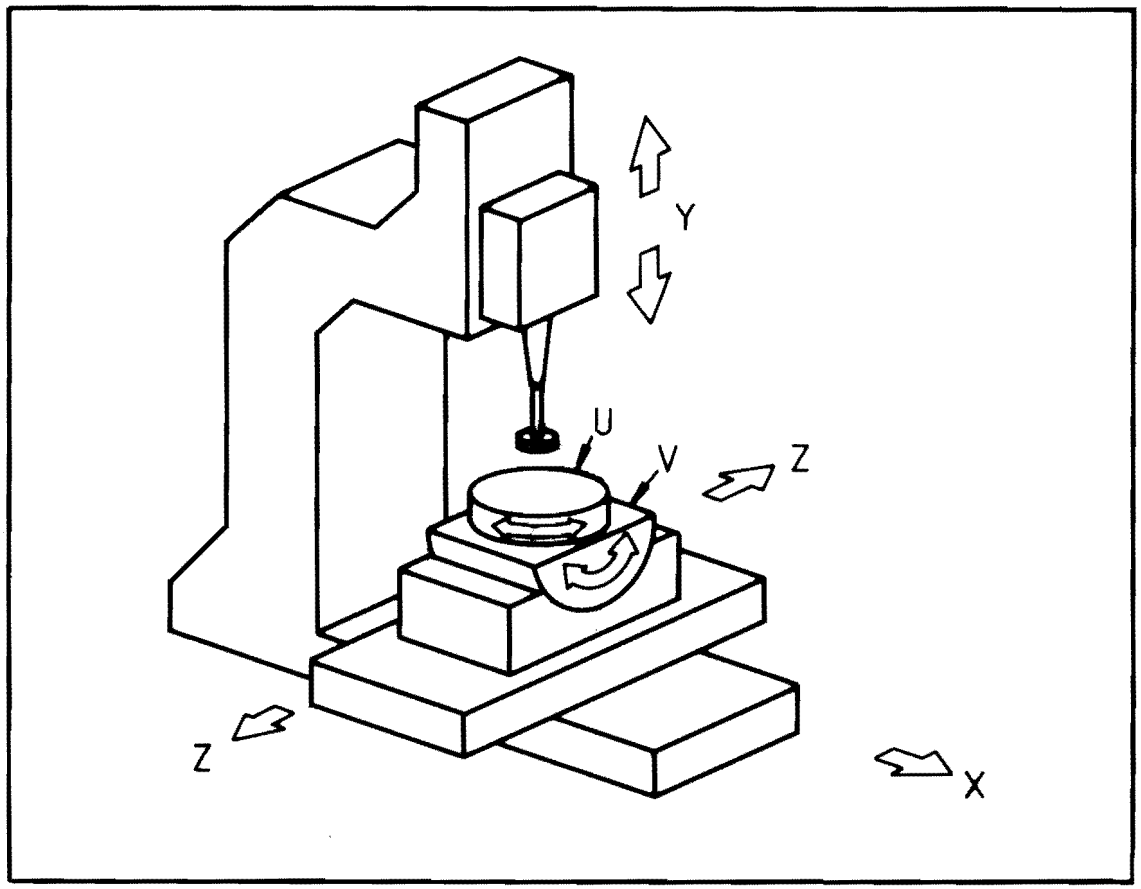


Figure 2.2 A Five Axis Milling Machine.

workpiece which has a certain number of degrees of freedom with respect to an observer.

(3) Stewart Machines: These machines control the six degrees of freedom of a platform by restraining it's D.O.F. by six independent links forming a complex closed kinematic chain. Each link or "leg", provides one constraint. Several different configurations are possible as is shown in Figure 2.3.

A present these machines are used in:

(a) A tyre testing machine to provide control of the tyre to road interface geometry,

(b) Aircraft flight simulators to provide simultaneous control of an aircraft cockpit simulator in attitude and acceleration,

(c) An automobile driving simulator similar to (b).

These machines are now briefly compared using these criteria:

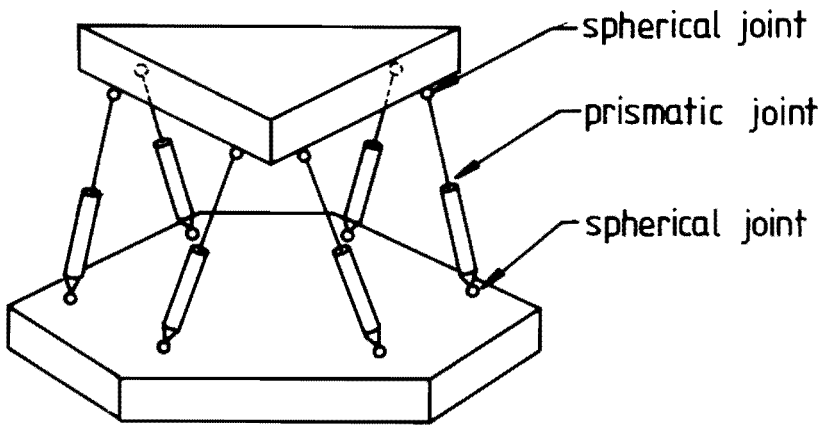
(a) Relative size of their envelope of movement,

(b) Potential speed of movement,

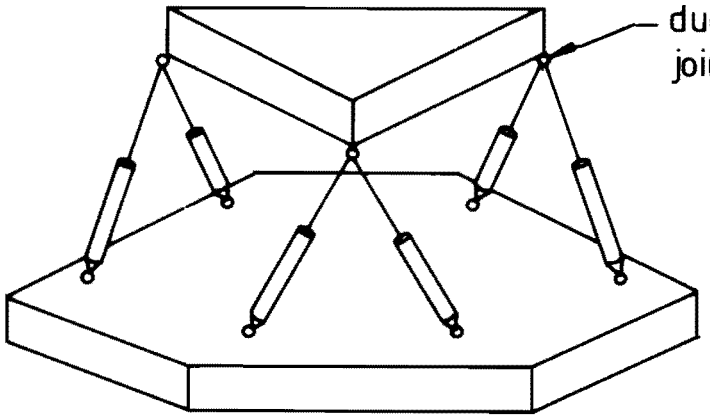
(c) Stiffness and strength,

(d) Ease of control.

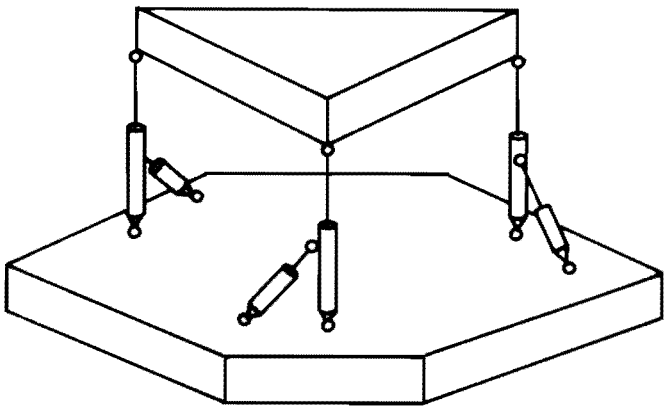
(1) Serial link open chain robots are known to have a large envelope of movement coupled with a high potential speed. They however have a limited stiffness due to the cantilever support of the controlled point, and require high speed computer algorithm solution for useful control, due to a complex dependence of the input variables



(a) CONSTRAINED TO SIX SPHERICAL SURFACES



(b) CONSTRAINED TO 3 ARCS



(c) CONSTRAINED TO 3 ARCS – LEGS TAKE BENDING MOMENT

Figure 2.3 Three Stewart Platform Configurations.

(usually joint angles) on the position of the controlled point. See [4].

(2) Milling machines have a small envelope of movement in relation to their size which is specially tailored for their application. They have a moderate speed capability with most of their operation occurring at very slow speeds.

They are specially constructed to provide a strong and stiff connection between the cutting head and the workpiece.

The actuator inputs to milling machines are usually independent or inter-dependent in a simple manner, so they are easy to control.

(3) Stewart machines have a relative envelope of movement similar to that of milling machines, and their speed capability too is similar. Strict comparisons are hard to make because the Stewart machine has a highly non-linear relation between platform and actuator movement, which is not conducive to direct analysis. Reference [1] gives details of forward and backward coordinate-geometry analyses.

Stewart machines are inherently stiff and strong because of their multiple closed kinematic chain structure. Their control requirements are simpler than those for serial link kinematic chain robots because the links are not serially coupled.

The above points are summarised in Table 2.1.

TABLE 2.1

Comparison of Leading Attributes of Six D.O.F. Machines

. = Poor Capability ..... = Excellent Capability

Attribute	Serial Link Robot Arm	Milling Machine	Stewart Machine
Relative envelope of movement	.....	..	..
Speed capability	.....	..	..
Stiffness	.	....	.....
Ease of control	.	....	..

2.2 Use of the Stewart Machine for Milling

From the above comparisons it appears the Stewart machine might be a competitor to the conventional 5 axis milling machine even though it has an unneeded extra D.O.F.. It appears to have equal or greater capabilities in all the criteria except ease of control. This last criterion however has minor importance because of the power, speed and low cost of digital computers.

There is however a previously unmentioned advantage of the Stewart machine over a conventional milling machine. This is the much greater mechanical simplicity of the actuators and links.

Whereas a typical milling machine comprises many varied components including rotary and linear bearings and support members, the Stewart machine in comparison has

only 3 link types; the base, the platform and the 6 identical supporting links (in the configuration shown in Figure 2.3a). The Stewart machine would therefore appear to have an advantage as regards cost of construction over a conventional milling machine, at no compromise in accuracy or stiffness capabilities. Refer to Chapter 8 for a cost estimate of a prototype Stewart Platform milling machine.

CHAPTER 3

Motion in Six Dimensions

3.1 Point to Point Motion

This chapter analyses the movement of a general platform in six space (symbolised by SS).

Six space is the geometric space in which a body can move with respect to a reference coordinate system.
A point C within this space can be described by:

$$C = (u,\ v,\ w,\ \phi,\ \theta,\ \psi) \qquad (3.1)$$

where u, v, w, describe the translations of the point with respect to the reference origin along the cartesian axes X, Y, Z of the reference system, and ϕ, θ, ψ describe the Euler rotations of the point with respect to the reference system axes. To enable this orientational specification, the point must be fixed to a movable coordinate system so that a local attitude of the point may be specified. The axes of this movable cartesian system are denoted by x, y, z.

We will thus consider the movements of a fixed point on the platform described by:

$$\underline{p} = (a,\ b,\ c) \qquad (3.2)$$

$\underline{p}$ could thus could represent a point on a workpiece or the cutting point of a tool attached to the platform.

A general movement of the platform might be described by:

"Move the point $\underline{p}$ on the platform from a starting position of C_S in SS to a finishing position of C_F in SS".

The geometric representation of this movement is shown in Figure 3.1. $\underline{p}$ is shown moving on a straight line from C_S at the upper left to C_F at lower right. Note that the orientation of $\underline{p}$ in the finishing position is different from that in its starting position. Therefore, not only the u, v, w, but also the ϕ, θ, ψ terms in C_S and C_F must differ in this particular movement.

Any point C in SS may be described by the following vectors, shown in Figure 3.1.

C - is the general six dimensional vector fully describing the position and orientation in six space of the object containing the vector $\underline{p}$.

$\underline{p}$ - as previously described is a fixed vector in the movable coordinate system

$\underline{T}$ - is a translation vector in the fixed coordinate system describing the movable coordinate system origin.

$\underline{P}$ - is a vector in the fixed coordinate system giving the spatial position of a point C in SS.

So from equation 3.1 the spatial position of C is given by:

$$\underline{P} = (u, v, w) \qquad (3.3)$$

Since the vector $\underline{p}$ can take any orientation in SS and yet still be at $\underline{P}$, then a point in three dimensions (X, Y, Z) can have an infinite number of points in SS.

$[R]\underline{p}$ - is a vector in the fixed reference system. $[R]\underline{p}$ can be described as:

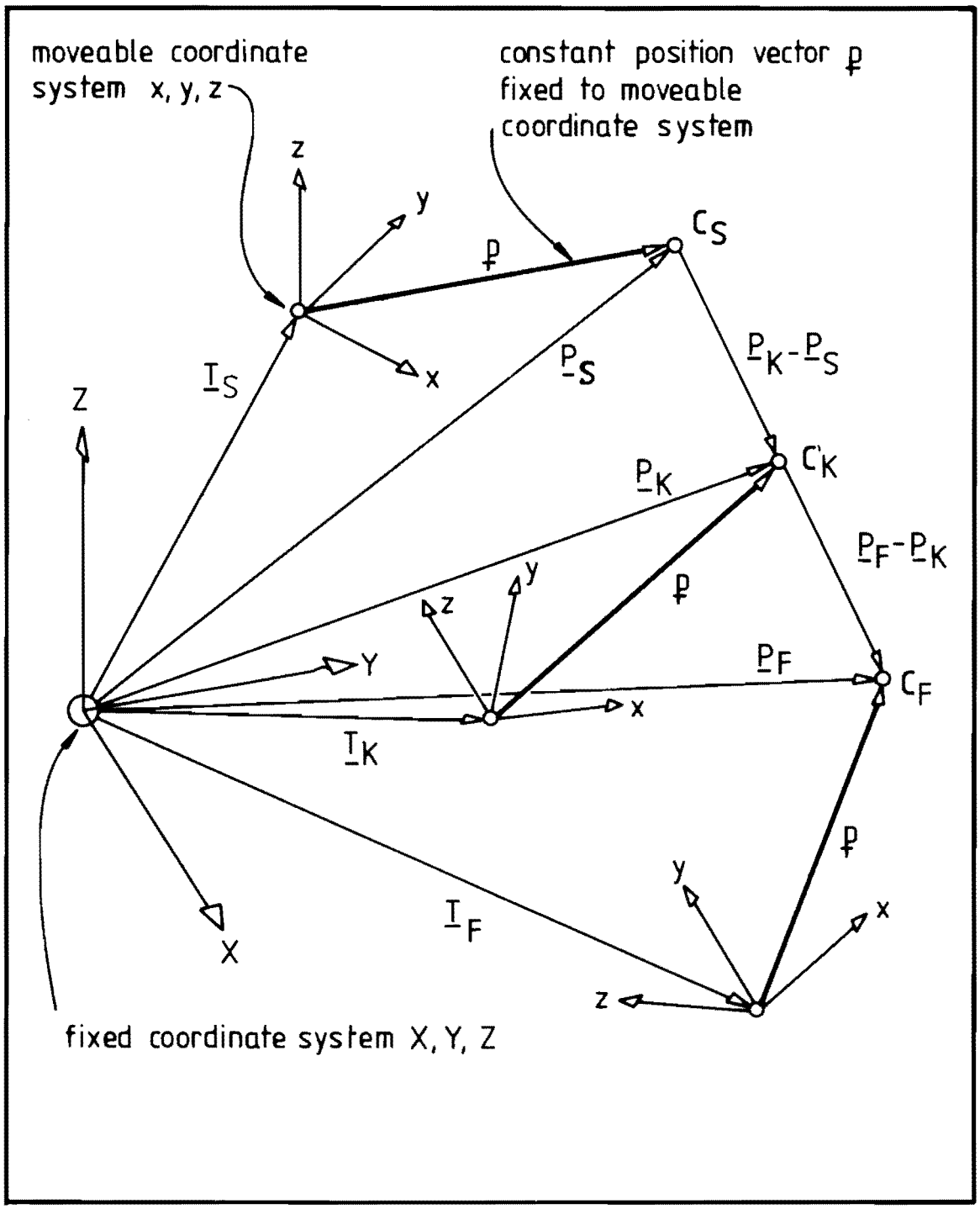


Figure 3.1 Representation of a general movement in six space.

The new orientation of the vector $\underline{p}$ after the specified Euler rotations have been applied, when the fixed and movable coordinates systems were originally coincident. This is shown in Figure 3.2.

In Figure 3.2, $\underline{P}_S$ is mapped onto the vector $\underline{P}_F$ by successive rotation of ϕ, θ, and ψ about the Z axis, then the new Y axis (Y2), then the new Z axis (Z1).

[R] is a 3x3 matrix operator and is a function of the Euler angles ϕ, θ and ψ contained in C.

The vector $\underline{p}$ can thus be moved to any orientation with respect to the fixed coordinate system by operating on it by the appropriate [R] matrix.

Thereafter, the vector $\underline{p}$ can be moved anywhere in three dimensions by adding it to an appropriate translation vector $\underline{T}$. It's orientation will remain unchanged.

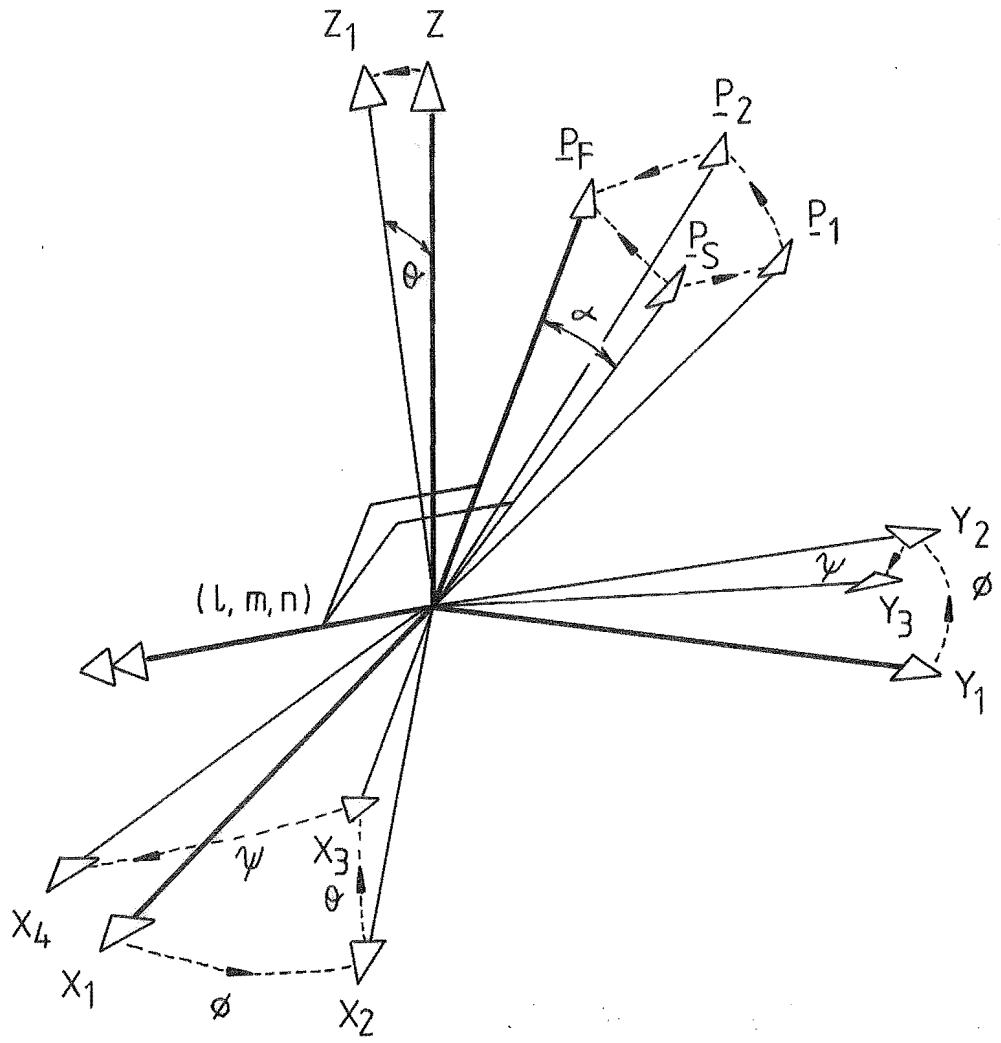
From Figure 3.1 it is apparent that

$$\underline{P} = \underline{T} + [R]\underline{p} \qquad (3.4)$$

or $$\underline{T} = \underline{P} - [R]\underline{p} \qquad (3.5)$$

For the platform to be brought to the correct position, $\underline{T}$ must be known, and equation 3.5 enables us to calculate $\underline{T}$ for any given C in SS and $\underline{p}$ in xyz.

Thus, using equation 3.5, we have sufficient information to position any fixed point $\underline{p}$ on the platform at any position $\underline{P}$ in our fixed reference system with any orientation of $\underline{p}$ (within the platform's envelope of movement). We therefore can achieve a "point-to-point" control from a starting state C_S to a finishing state C_F.



$\underline{P}_S$ can change orientation to $\underline{P}_F$ by either:-

a) Euler rotations of ϕ, θ then ψ

b) single rotation of α about line (l, m, n)

Figure 3.2 Two methods of specifying orientation changes

3.2 Path Control

Any control of the path of a numerically controlled positioning machine implies the automatic generation of points on a specified path between starting and finishing positions. When no other information is given, it is assumed the path is straight, and that any orientation changes are in a uniform direction. Such path control will be called "basic path control".

This point generation is not needed on some axes of certain machines (mainly milling machines) because the actuator position is proportional to the geometric position, so "point to point" control automatically provides the aforementioned basic path control.

Any other automatic path control (i.e. circular) can be achieved by composing the desired path from many short paths each having basic path control, or by using a different automatic path generation method of the required type.

We wish to derive the geometric requirements that will enable the automatic generation of intermediate points for basic path control for a point $\underline{p}$ on a movable platform moving from a starting state C_S to a finishing state C_F in six space.

We assume the movement will be divided into N geometrically equal parts. The value of N is determined from considerations of the required accuracy of the movement, and the kinematics of the mechanism. For example, N will equal 1 (trivial case) for a movement of the X cross-slide shown in Figure 2.2 because the machine

kinematics automatically assure basic path control in this direction. On the other hand, a Stewart machine will require at the least N = 1 and for large movements and/or when the required path control accuracy is high, a large value of N. This is because the relation between actuator and geometric states for such a machine is highly non-linear. This non-linearity also creates a difficulty in calculating the required value of N to achieve a specified degree of path control.

So, referring again to Figure 3.1 , we wish to move from a starting state of C_S to a finishing state C_F under basic path control.

The geometric analysis of the movement can be split into positional and rotational interpolation analyses.

3.2.1 Positional Interpolation

The three dimensional path from C_S to C_F must be on a straight line and divided into N equal increments.

Thus, for the k'th point on the line we require that

$$\underline{P}_K = \underline{P}_S + k/N(\underline{P}_F - \underline{P}_S) \qquad (3.6)$$

so that when k = N,

$$\underline{P}_K = \underline{P}_S + (\underline{P}_F - \underline{P}_S) = \underline{P}_F$$

and when k = 0,

$$\underline{P}_K = \underline{P}_S$$

3.2.2 Rotational Interpolation

The orientation of $\underline{p}$ in the reference coordinate system must change from its starting state to it's finishing state in N equal geometric movements. The "equal" condition requires that the attitude changes be of equal magnitude and about an axis of rotation maintaining a constant direction.

We can analyse the changes of the orientation of $\underline{p}$ by assuming the fixed and movable coordinate systems were originally coincident, because a translation $\underline{T}$ can be subsequently applied to a rotated $\underline{p}$ to position $\underline{p}$ to any specified point C in SS.

That is, the attitude of $\underline{p}$ and the position vector $\underline{P}$ are independent.

Let the starting orientation of $\underline{p}$ be described by:

$$\underline{P}_S = [R]_S \underline{p} \qquad (3.7)$$

and the finishing orientation of $\underline{p}$ be described by:

$$\underline{P}_F = [R]_F \underline{p} \qquad (3.8)$$

and an intermediate orientation, the k'th attitude be described by:

$$\underline{P}_K = [R]_K \underline{p} \qquad (3.9)$$

We wish to obtain an expression for $[R]_K$ involving $[R]_S$, $[R]_F$ and the ratio k/N to obtain an intermediate point on the circular path from $\underline{P}_S$ to $\underline{P}_F$.

Now,

$$\underline{p} = [R]_S^T [R]_S \underline{p} \qquad (3.10)$$

Thus equation 3.8 can be expressed as:

$$\underline{P}_F = [R]_F [R]_S^T [R]_S \underline{p}$$

or,

$$\underline{P}_F = [R]_{SF}[R]_S\underline{p} \qquad (3.11)$$

where

$$[R]_{SF} = [R]_F[R]_S^T \qquad (3.12)$$

Thus,

$$[R]_F = [R]_{SF}[R]_S \qquad (3.13)$$

$[R]_{SF}$ corresponds to the attitude functional operator acting on the vector:

$$\underline{P}_S = [R]_S\underline{p}$$

to create a finishing vector

$$\underline{P}_F = [R]_{SF}([R]_S\underline{p})$$

Terms of ϕ, θ, and ψ could be extracted out of $[R]_{SF}$ that would describe the Euler rotations necessary to map $\underline{P}_S$ onto $\underline{P}_F$. Such rotations do not occur about a constant axis of rotation however, which is what we wish to achieve.

$[R]_{SF}$ though can be re-formulated into an equivalent matrix, the Euler matrix, whose parameters specify the movements as being a rotation of α degrees about a line given by the direction cosines l, m, n. These equivalent movements are shown on Figure 3.2.

Thus, by using the variables l, m, n and α we could create an attitude operator that would have a constant direction (by keeping l, m, n constant) but a variable degree of rotation (by varying α).

So, to achieve a direct path from $\underline{P}_S$ to $\underline{P}_F$ we can operate on $[R]_S\underline{p}$ by $[R]_{SF}$:

(a) where $[R]_{SF}$ = FUNCTION1(ϕ, θ, ψ) (3.14)

(see Appendix 1) and the movement is equivalent to movement (a) in Figure 3.2, or,

(b) where $[R]_{SF}$ = FUNCTION2(l, m, n, α) (3.15)

(See Appendix 1) and the movement is equivalent to movement (b) in Figure 3.2.

Using the above function and arguments of choice (b) we can then define the k'th intermediate point between $\underline{P}_S$ and $\underline{P}_F$ by using:

$$[R]_{SF}^{k/N} \equiv \text{FUNCTION2}(l, m, n, \frac{k}{N}\alpha) \quad (3.16)$$

so that,

$$\underline{P}_K = [R]_{SF}^{k/N}[R]_S\underline{p} \quad (3.17)$$

When k=0,

$$[R]_{SF}^{0/N} = I \quad \text{(from 3.16)}$$

so,

$$\underline{P}_0 = [R]_S\underline{p} = \underline{P}_S$$

and when k=N,

$$[R]_{SF}^{N/N} = [R]_{SF}$$

so,

$$\underline{P}_N = [R]_{SF}[R]_S\underline{p} = [R]_F\underline{p} \quad \text{(from 3.12)}$$

Thus, the desired attitude function operator $[R]_K$ for equation 3.9 is:

$$[R]_K = [R]_{SF}^{k/N}[R]_S \quad (3.18)$$

where $[R]_{SF}^{k/N}$ is defined from equations 3.12 and 3.16.

3.2.3 Summary

To completely specify the position of the platform we need to know the translation vector $\underline{T}$. This is obtained by using the relations derived from equations 3.6 and 3.18 in equation 3.5.

These are:

(a) Positional interpolation requirement:

$$\underline{P}_K = \underline{P}_S + k/N(\underline{P}_F - \underline{P}_S) \qquad (3.6)$$

(b) Rotational interpolation requirement:

$$[R]_K = [R]_{SF}{}^{k/N}[R]_S \qquad (3.18)$$

(c) Platform origin specification:

$$\underline{T}_K = \underline{P}_K - [R]_K\underline{p} \qquad (3.19)$$

(from 3.5)

Giving:

$$\underline{T}_K = \underline{P}_S + k/N(\underline{P}_F - \underline{P}_S) - [R]_{SF}{}^{k/N}[R]_S\underline{p} \qquad (3.20)$$

Thus, using equations 3.6, 3.18 and 3.20 a complete specification of the basic path control of an object in six space may be obtained.

The calculation procedure is shown in Figure 3.3 for a commanded movement of $\underline{p}$ from C_S to C_F with a resolution of R at a speed V. This figure shows the generation of N+1 data sets, each set consisting of machine independent positional and speed information. Each of these sets is then converted into a corresponding set of machine dependent actuator state vectors which can be used to control a machine.

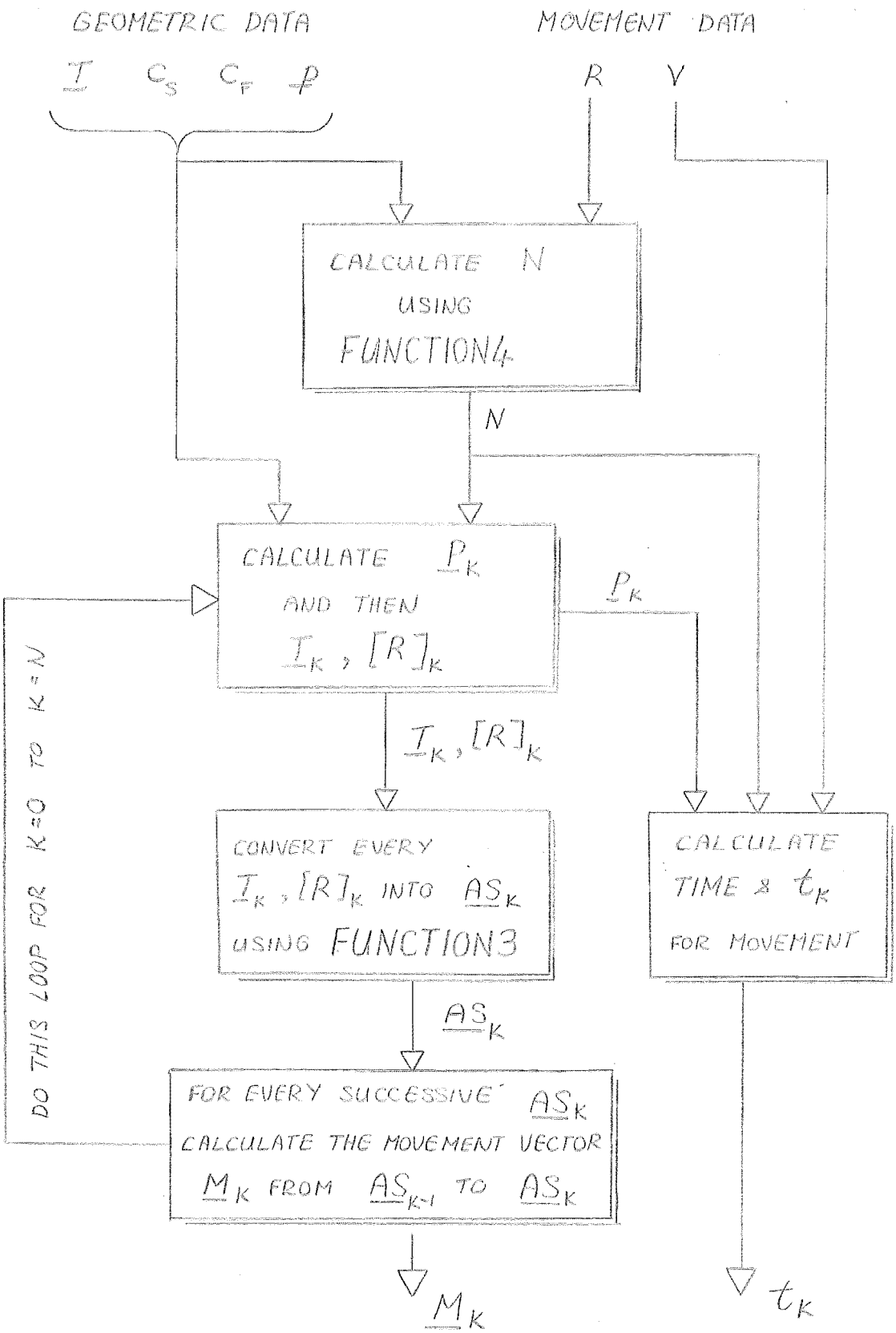


Figure 3.3 Path Control calculation procedure.

3.3 Path Synthesis Using the UoC Stewart Machine

Adaptation of the path synthesis method shown in Figure 3.3 for the Stewart machine consists of creating the functions FUNCTION3 and FUNCTION4 for this particular machine. The creation of these functions is detailed below.

FUNCTION3

FUNCTION3 is a composite of 2 functions, denoted by

$$\text{FUNCTION3} = \text{FUNCTION3B}(\text{FUNCTION3A}) \qquad (3.21)$$

FUNCTION3A was developed in [1] pages 98-103 and converts the machine independent position data ($\underline{T}_K$, $[R]_K$) into machine dependent geometric data $\underline{l}_K$, where $\underline{l}_K$ represents the six element leg length vector that will create the commanded platform state.

$$\text{i.e.} \quad \underline{l}_K = \text{FUNCTION3A}(\underline{T}_K, [R]_K) \qquad (3.22)$$

where $\underline{l}_K = (l_1, l_2, l_3, l_4, l_5, l_6)_K$

The significance of some of these previously mentioned variables is shown in a geometric representation of the UoC Stewart platform shown in Figure 3.4.

The six legs of the table are lead screws actuated by stepper motors via end mounted Hooke joints. Thus, the leg length vectors $\underline{l}_K$ must be converted into appropriate stepper motor actuator state vectors, denoted by $\underline{AS}_K$. The units of these vectors are angular steps.

This conversion is done by FUNCTION3B which is derived below.

For any leg l_i, the relation between length and steps is:

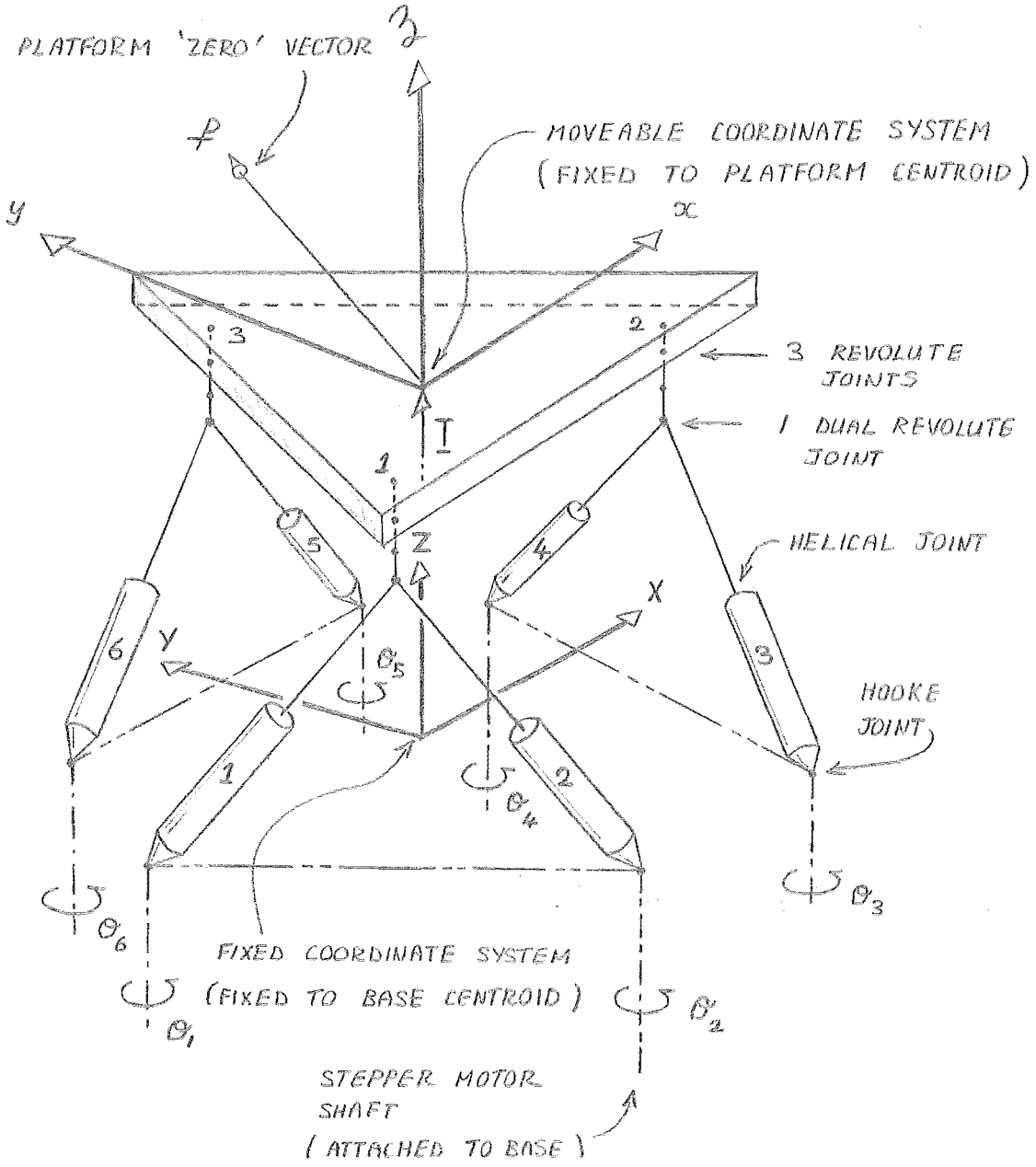


Figure 3.4 The UoC Stewart platform geometry.

$$l_i = \text{PITCH}[(N_i + N_Z)\theta_D + \theta_H] \qquad (3.23)$$

where l_i = length of leg i in mm,

θ_D = unit step angle of the motors (1.8 degrees),

N_i = actuator step state corresponding to l_i,

θ_H = cyclic angular error caused by the Hooke joint (degrees),

N_Z = "zero" length of the leg in steps,

PITCH = lead of the helix = 2.54mm/360 degrees

θ_H was assumed to equal zero to simplify the software. It's affect is limited to the maximum angular error of a Hooke joint which for shafts with a 45° inclination is +/- 9 degrees. This imposes a maximum cyclic error on the leg lengths of about +/- 0.06mm (about 5 steps).

Thus FUNCTION3B is (assuming a perfect universal joint):

$$N_i = \frac{l_i}{\text{PITCH}.\theta_D} \qquad (3.24)$$

and this function converts every length vector into a corresponding actuator vector i.e.:

$$\underline{AS}_K = \text{FUNCTION3B}(\underline{l}_K) \qquad (3.25)$$

where $\underline{AS}_K = (N_1, N_2, N_3, N_4, N_5, N_6)_K$

FUNCTION3 may be found in the "LENGTH" subroutine of the computer listing (Appendix 2). It takes approximately 1/2 second to calculate each $\underline{AS}_K$ from the input $\underline{T}_K$ and $[R]_K$.

FUNCTION 4

Another function that is derived from the machine geometry is FUNCTION4 shown on Figure 3.3. FUNCTION4 seeks to determine the minimum value of N that will ensure the motion of $\underline{p}$ will follow the required path from C_S to C_F within a "tunnel of resolution" of radius R. The 'perfect' derivation of this function was not attempted for these reasons:

(a) The derivation of FUNCTION4 was not at all obvious and if derived would probably involve repeated use of the inverse of FUNCTION3A, know to be in itself complex. (see [1], pages 105-109)

(b) It is not necessary to use a minimum value for N. A larger value of N can be tolerated. The cost being in computation and storage of the extra geometric and actuator vectors.

Thus, a simple approximation to FUNCTION4 was required that would always give a value of N greater than the ideal value.

This function was derived by considering the error involved in a motion such as shown in Figure 3.5. This diagram shows the platform moving a distance m from $\underline{P}_S$ to $\underline{P}_F$ so that one leg has the same starting and finishing length. If this movement is made in 1 part, then the error in movement will be e, as the leg's length will remain the same as it swings from $\underline{P}_S$ to $\underline{P}_F$.

The relation between e and m is:

$$m = \sqrt{8le} \qquad (3.26)$$

assuming

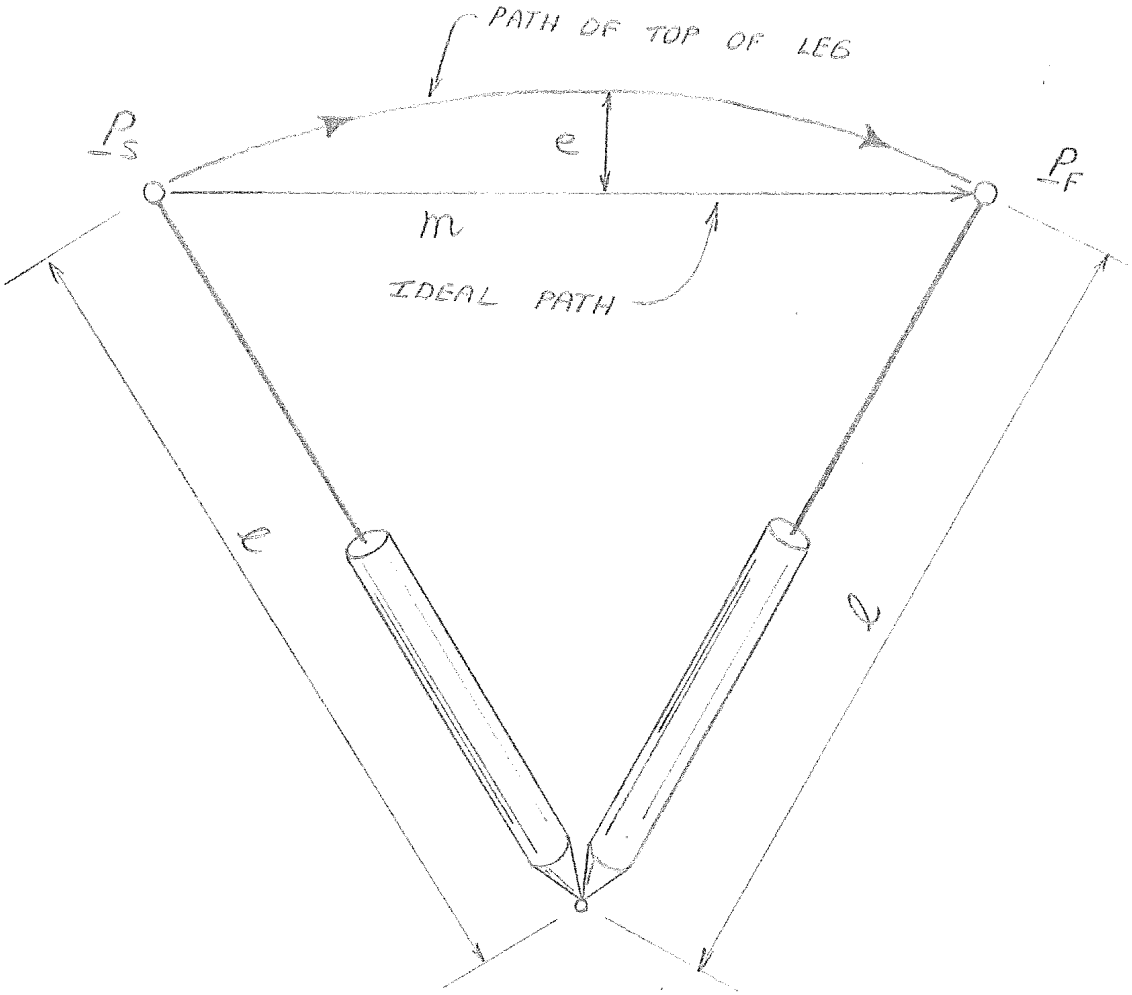


Figure 3.5 Geometry of path deviation.

$e << 2l$

While this relation may be true for this leg, it will not in general be true for the other 5 legs. However, the other legs will undergo a similar type of "rocking" movement and so their errors would be of a similar magnitude.

It is assumed that the maximum error on any leg will be less than four times the error of this special case for any direction of movement. Thus, if we put the allowable error as e/4, then the maximum error should not exceed e. Thus, the allowable movement m becomes:

$$m = \sqrt{2le} \qquad (3.27)$$

This relation yields for l = 250:

Table 3.1

Typical Allowable Moves m for Maximum Error of e

Path Error e (mm)	Allowable Movement m (mm)	Approx. Actuator Movement (steps)
0.001	0.7	39
0.01	2.2	122
0.1	7.1	395
1.0	22.4	1247
10.0	70.7	3936

Equation 3.27 allows us to calculate how many parts N any movement M must be split into to be assured that the path will not deviate by more than e from the desired path under "basic path control".

Thus FUNCTION4 is:

$$N = \frac{|M|}{m} \qquad (3.28)$$

where $m = \sqrt{2le}$

and l is taken as a constant equal to the minimum leg length.

For purely translational movements:

$$M = \underline{P}_F - \underline{P}_S \qquad (3.29)$$

For rotational movements, the movement of a corner of the platform is used where:

$$M = 100\ \alpha \qquad (3.30)$$

where α is the angular rotation in radians.

For combined movements, the maximum value of equations 3.29 and 3.30 is used.

3.4 Conclusion

This chapter has developed a method for translating a six dimension path control specification into the required actuator data that will generate the required path for any particular machine. The method is shown in Figure 3.3.

The method has been developed for the University of Canterbury Stewart machine by creating the machine specific functions; FUNCTION3 and FUNCTION4.

CHAPTER 4

Multiple Open-Loop Stepper Motor Control

Figure 4.1 shows a simplified schematic diagram of the control system used to simultaneously control the 6 stepper motors that activate the Stewart Platform.

The control hardware and theory developed in this chapter can be applied to other systems having a different number of motors. All the following discussion however refers solely to the Stewart Platform application.

Note that the whole system is split into three parts.

(a) The six Actuator Control Units (ACUs),

(b) The Machine Control Unit (MCU),

(c) The Computer Control System (CCS).

Briefly, the computer control system controls the MCU which simultaneously controls all 6 ACUs. The computer cannot simultaneously control 6 ACUs, hence the need for the MCU.

The computer control system, based on the "CCS" program uses a data file for the particular motion to control the MCU which in turn controls the ACUs.

The ACUs and the MCU have special characteristics which determine how they are controlled by the computer. These characteristics affect the form of the "COMPILE" program which creates the data file that "CCS" uses, and the "CCS" program itself. These characteristics are now discussed in turn.

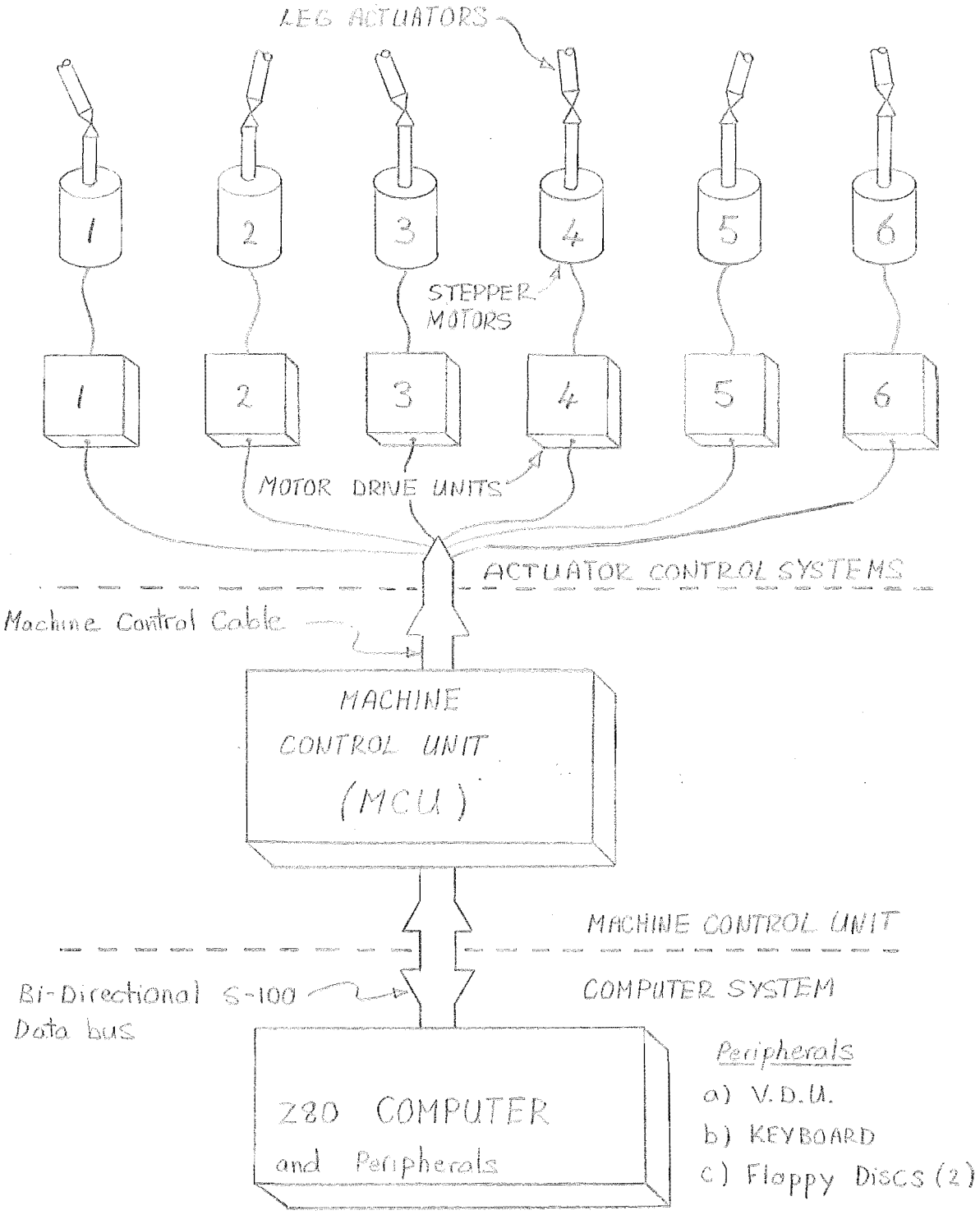


Figure 4.1 Stewart platform control system schematic.

4.1 The Actuator Control Units

The actuator control unit for a leg "i" is shown schematically in Figure 4.2. This is duplicated for every other leg.

ACU Input:

(a) P_i : Pulses on the position lines P_i. Each pulse commands the stepper motor to increment it's position 1 step in the direction determined by the D_i line.

(b) D_i : A direction controlling logic level on the direction lines D_i. A logic high causes pulses to create clockwise increments, a logic low causes pulses to create anti-clockwise increments.

ACU Output:

θ_i : The angular position of the stepper motor shaft.

ACU Operational Function:

The stepper motor shaft position at time t is the sum of all the preceeding step increments, with each increment either adding or subtracting from the total depending on the value of the direction command when the increment-causing pulse arrived. That is:

$$\theta_{it} = \theta_0 + a \sum_{k=0,\, P_{it}} D_{ik} \qquad (4.1)$$

Where θ_0 - is the starting position of the shaft,

a - is the constant angular increment of the stepper motor (= 1.8 degrees),

P_{it} is the number of incrementing pulses

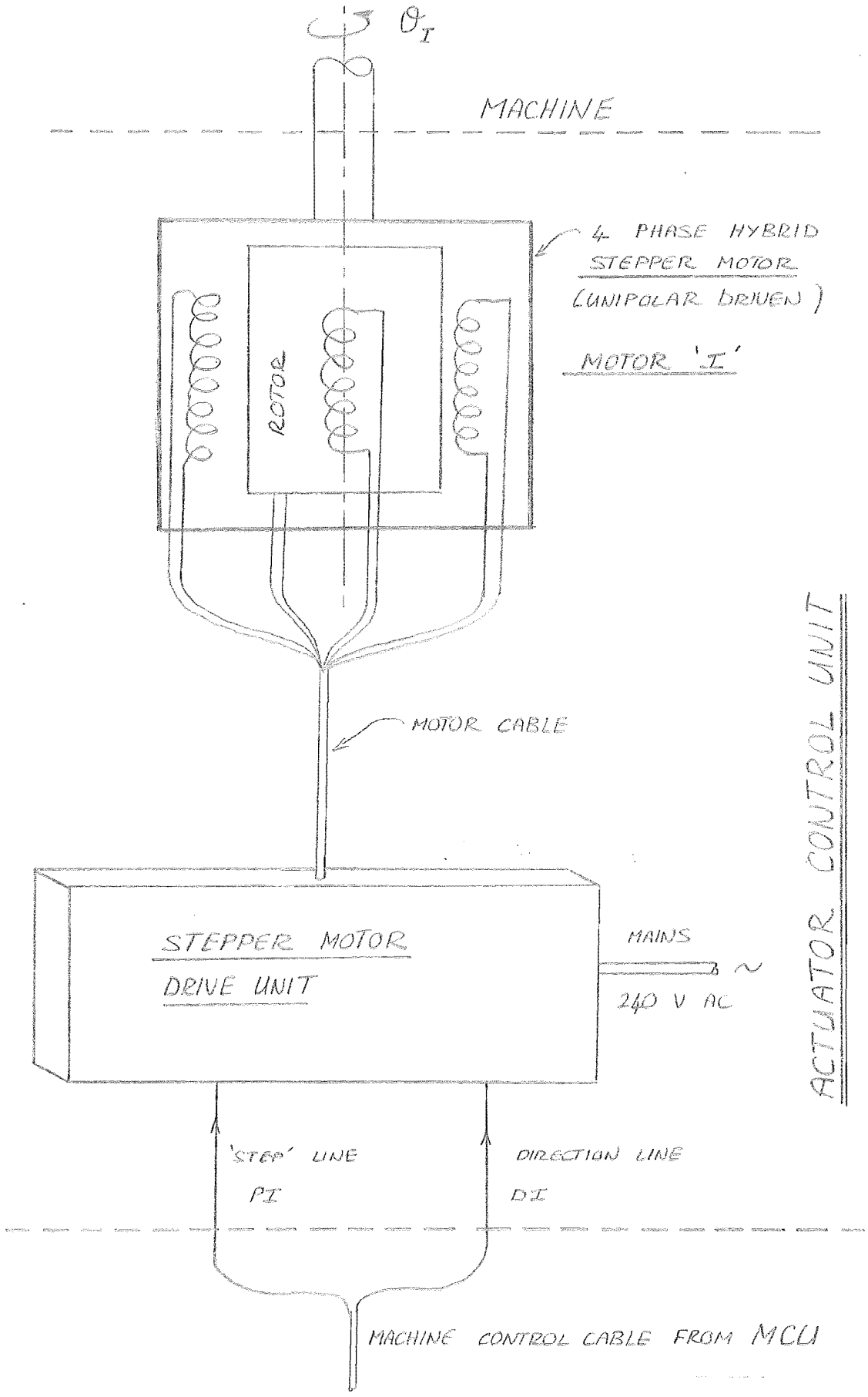


Figure 4.2 Actuator Control Unit schematic.

received from the starting time, to the time t,

D_{ik} is the value of the direction control command when each increment pulse arrives. D_{ik} = +1 for a clockwise rotation, D_{ik} = -1 for an anticlockwise rotation.

ACU Characteristics

The system ideally follows equation 4.1 except for the affects of rotor inertia and load on the magnetic coupling of the stepper motor.

The performance of a motor is usually presented in the form of a graph as shown in Figure 4.3. See [5] for more explanation.

On this graph the two curves represent the performance of a motor having a zero value of load inertia.

The "pull in curve" represents the torques available at speeds that the motor/load combination can immediately attain from rest.

The "pull out curve" represents the maximum torque available at a constant speed without the motor "losing" any steps.

Reliable operation in the "slew" region between these two curves is dependent on the commanded acceleration. If an acceleration is attempted of greater magnitude than the motor/load combination allows then the movement of the shaft will "miss" some of the electrical driving pulses being output. This will result in the shaft's angular position not being related to the number of pulses output to the motor.

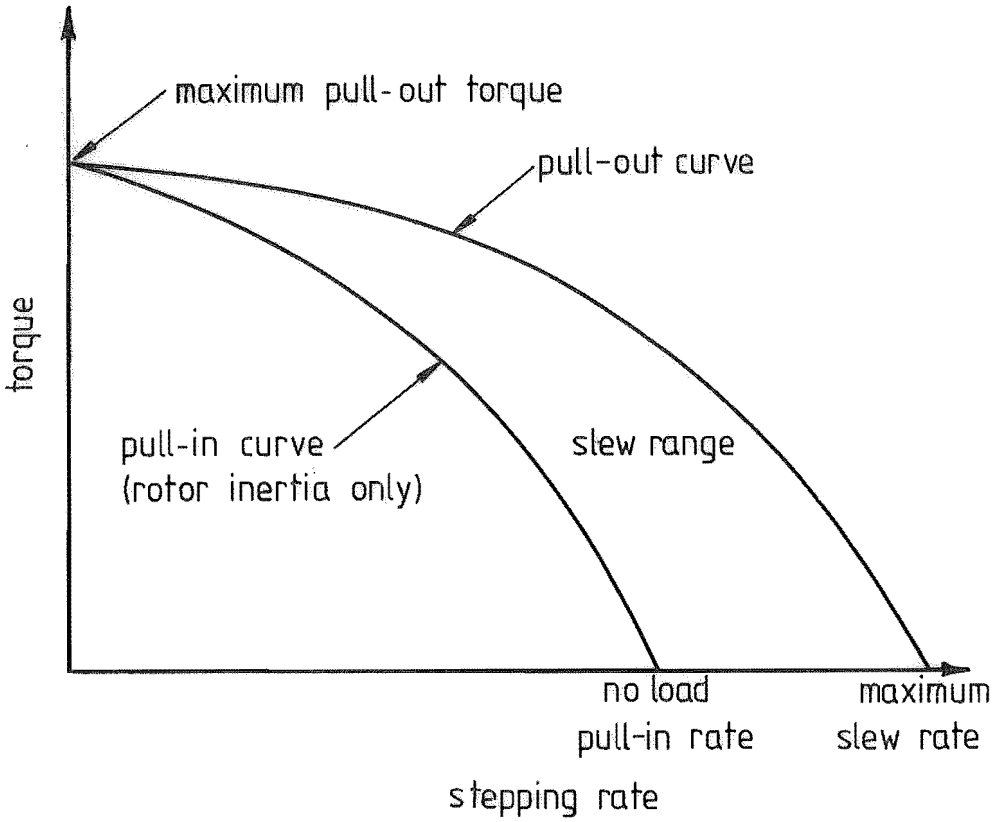


Figure 4.3 Typical stepper motor performance curves.

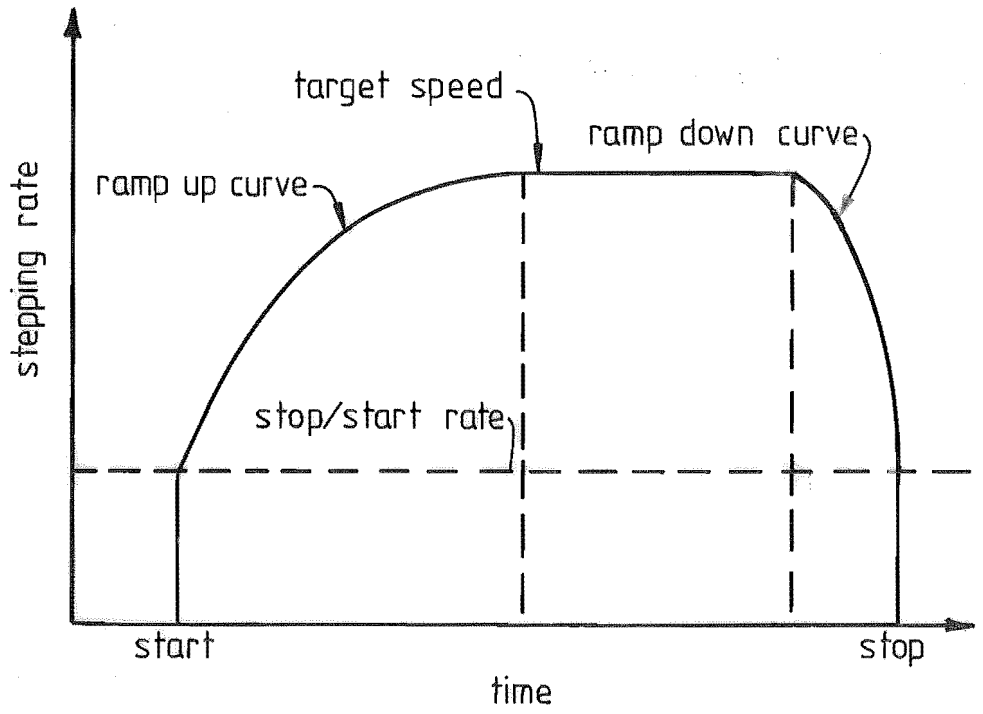


Figure 4.4 Typical stepper motor dynamic response.

The "pull in curve" of Figure 4.3 serves merely as a guide to the pull in speed of our actuators because the torque loading and inertia of our actuators are at present unknown.

Also, the allowable acceleration in the "slew range" is unknown for our motors. The maximum acceleration between any two velocities in the "slew region" will follow an acceleration profile as shown in Figure 4.4. The true profile can be determined by solution of the differential equation describing the stepper motor/load characteristics.

Summary

Each actuator control unit acts in a "digital" manner to the input signals P_i and D_i providing that the commanded accelerations to the actuator do not exceed a certain value that is a function of speed, torque and inertia as show in Figures 4.3 and 4.4.

4.2 The Machine Control Unit (MCU)

The MCU serves as an "intelligent" interface between the computer's S-100 data bus and the actuator control system's machine control cable. The MCU works in a "block by block" mode, converting a block of data sent from the computer on the S-100 bus, into the appropriate signals on the machine control cable. The actual operations that the MCU performs are :

(a) The MCU creates six pulse signals on the position lines P1 to P6 according to the speed data in the block,

(b) The MCU creates logic levels on the direction lines, D1 to D6 according to the direction data in the block,

(c) The MCU counts the number of steps output on each position line and sends an "interrupt" signal to the computer when the correct number of steps has been output, according to movement data in the block.

The structure of the MCU and it's main input and output signals is shown in Figure 4.5. It's layout is based on the theory given in reference [7].

4.2.1 MCU Operation

How the MCU converts the input data in the data block into signals on the position and direction wires is described below, referring to Figure 4.5.

(a) Creating Direction Controlling Logic Levels

The data is contained in the 8-bit word DIRG. At the start of motion at t = 0, this word is sent from the CCS and is "latched" to an 8-bit flip-flop. The word is thus held on the 8 flip-flop output pins. The lower 6 bits are connected to the 6 direction lines, creating logic levels on these wires equal to the logic levels on the lower 6 bits of DIRG. The remaining 2 bits are used for gating of other signals,

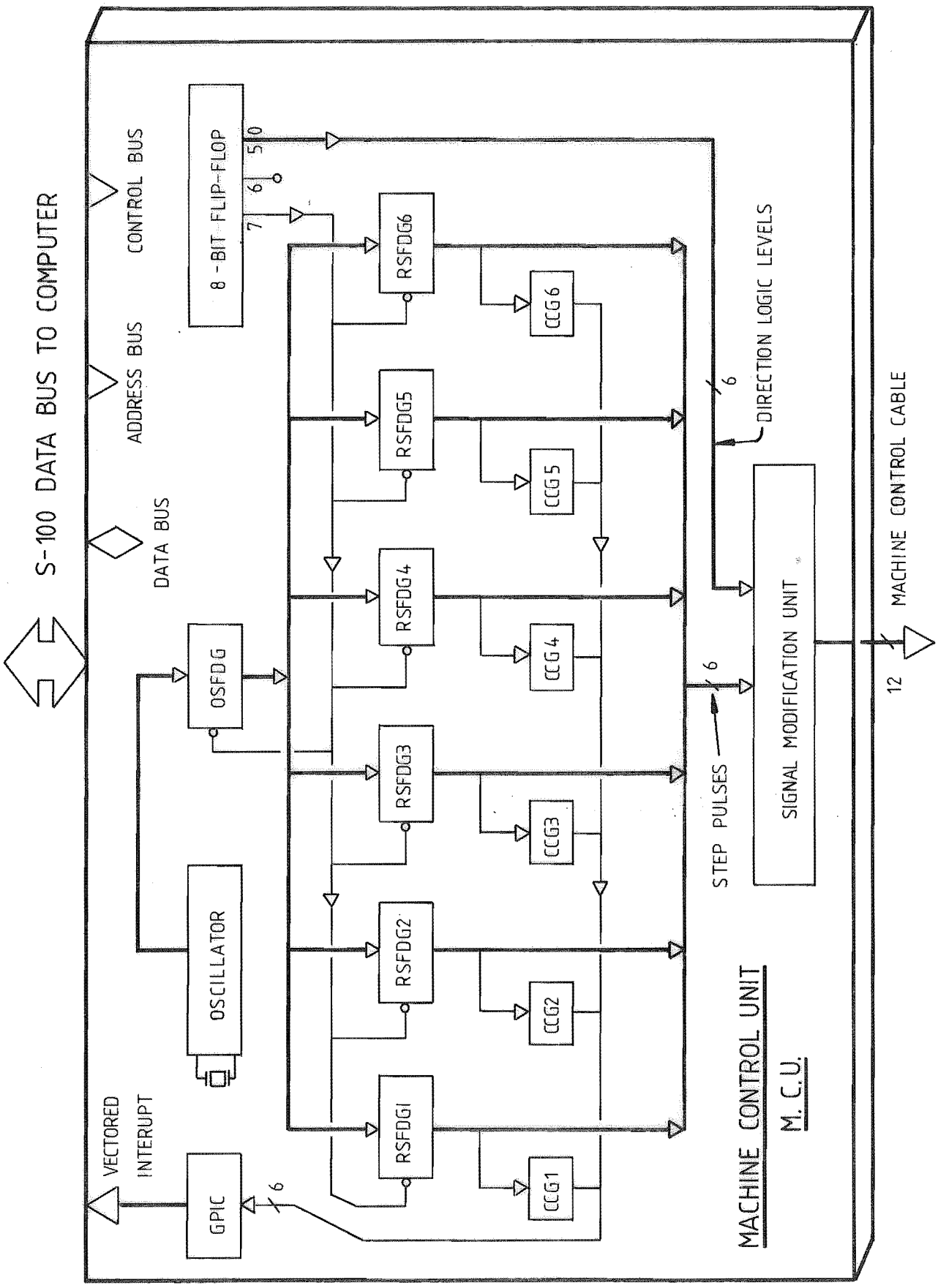


Figure 4.5 M.C.U. operational schematic.

hence the derivation of the DIRG description of the word from DIRection and Gating.

At t = T at the end of the block's motion a new DIRG word will be latched.

(b) Creation of Step Pulses

Firstly, six equal and synchronous pulse signals are created.

This is done using a fixed frequency oscillator producing a frequency of 4.9152 MHz. This signal is fed through a frequency divider called the "Overall Speed Frequency Divider Group" denoted by OSFDG. This group enables the frequency to be divided by any integer from 3 to 65535. This divisor is called the "Speed Divisor", denoted by SDIV. The operation of this frequency divider is shown in Figure 4.6(a).

The output from the OSFDG is used to feed 6 "Relative Speed Frequency Divider" Groups denoted by RSFDG1 to RSFDG6, shown in figure 4.6(b). These groups can divide the input frequency by any integer from 3 to 65536. (Division by 2 was found to produce erratic operation) These frequency divisors are called "Relative Speed Divisors", denoted by RSD1 to RSD6. These divisors control the relative speeds of the pulse signals being output from the six RSFDGs. These signals are then connected directly to the corresponding position control lines P1 to P6 on the machine control cable.

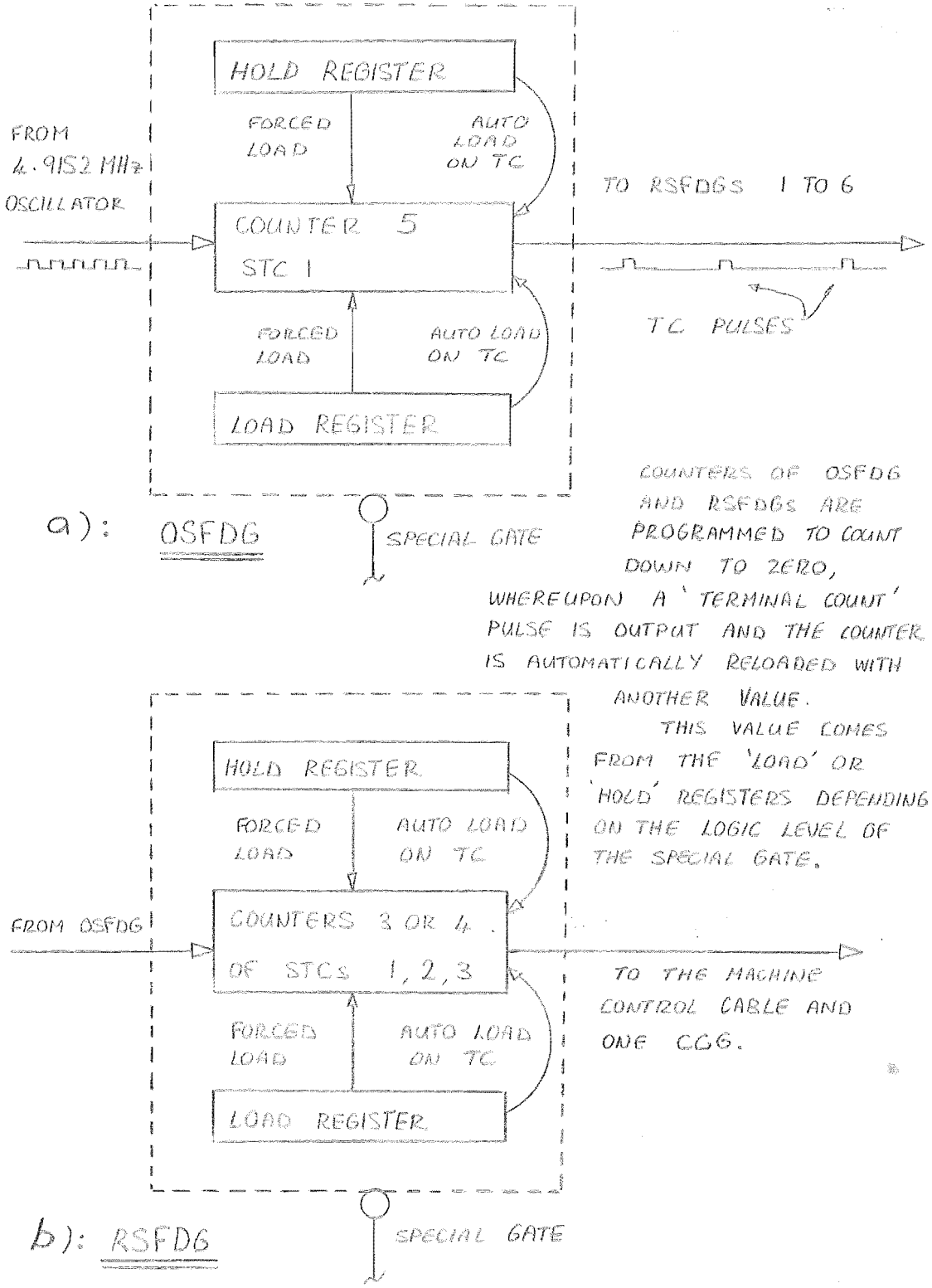


Figure 4.6 a) OSFDG operation.

Figure 4.6 b) RSFDG operation.

The speed of pulses on any position line P_i is thus:

$$V_i = \frac{F}{SDIV \cdot RSD_i} \qquad (4.2)$$

where F = oscillator frequency (= 4.9152 MHz).

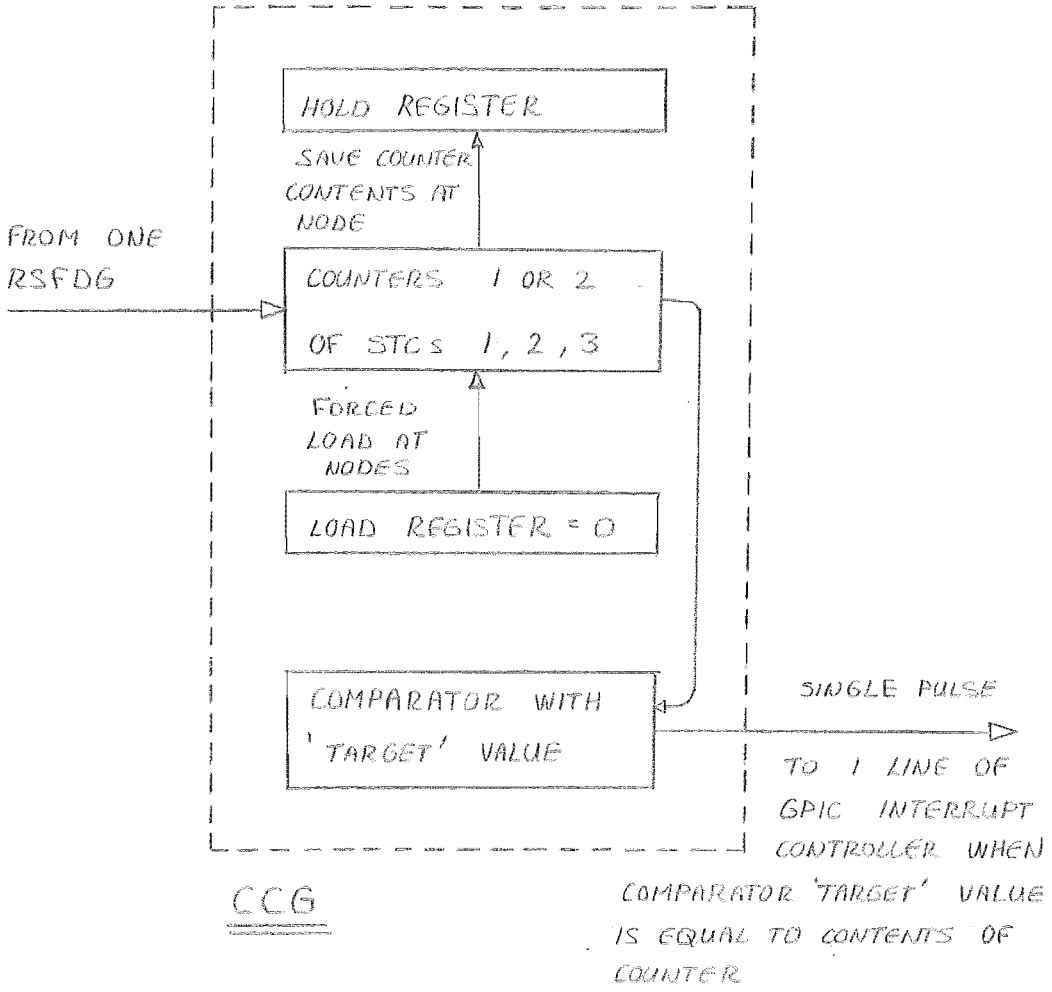
So, the speed data in the block, namely the 16-bit divisors SDIV, and RSD1 to RSD6, give an overall control over all the position lines' pulse rates via SDIV, and an individual control of each position line's pulse rate via each RSD_i.

(c) Monitoring the Number of Pulses Output

The pulse signals on each position line are counted by 16-bit "Counter Comparator Groups" denoted by CCG1 to CCG6 shown in Figure 4.7.

The contents of each CCG counter are initialised to zero at the start of every data block's movement. Thus, as the movement progresses, these groups hold the number of pulses output on each position wire.

For each data block, the comparator on the fastest motor is loaded with the correct number of steps to be output in that block, denoted by TARGET. When the counter contents equals the value in this comparator a pulse will be output from this comparator. This signals that the correct number of steps have been output to the fastest motor. This signal will be relayed to the Computer Control System via one channel of an 8 channel interrupt controller (GPIC) on the MCU.



THE COUNTER OF EACH CCG GROUP IS PROGRAMMED TO COUNT UP. IT'S CONTENTS ARE SAVED AND THEN SET TO ZERO BY SOFTWARE COMMANDS AT EACH SYSTEM NODE.

Figure 4.7 CCG operation.

This signal will inform the CCS that the movement for this block has been achieved and the CCS will initiate an almost instantaneous reload of a new data block to continue the movement.

The 6 relative speed divisors have been chosen so that the last pulses on each of the position lines occur at the same time (or just before) as the last pulse on the fastest motor position wire. Thus, arrival of the last pulse for the fastest motor implies that the last pulses for all the other motors have also just arrived.

The fastest motor is used to determine the end of the block movement to obtain the maximum temporal resolution.

The machine control data block (MCDB) composition is shown below:

BLOCK	2 bytes
INTMSK	1 byte
DIRG	1 byte
TARGET	2 bytes
RSD1	2 bytes
RSD2	2 bytes
RSD3	2 bytes
RSD4	2 bytes
RSD5	2 bytes
RSD6	2 bytes
SDIV	2 bytes

The function of the other elements in the data block is briefly explained:

(a) BLOCK: This word enables each data block to be identified by the CCS,

(b) INTMSK: This is an 8-bit mask that is sent to control the interrupt controller. It is used to "mask" any unwanted signals arriving at the interrupt controller. That is, all but the 1 signal from the CCG of the fastest motor.

4.2.2 Hardware Description

The pulse generation and manipulation tasks are performed using 3 VLSI chips called System Timing Controllers (3x AM9513s, see [8]). These chips provide a versatile means of tailoring many frequency manipulation tasks.

The interrupt controller chip used is the AM9519A (see [8]).

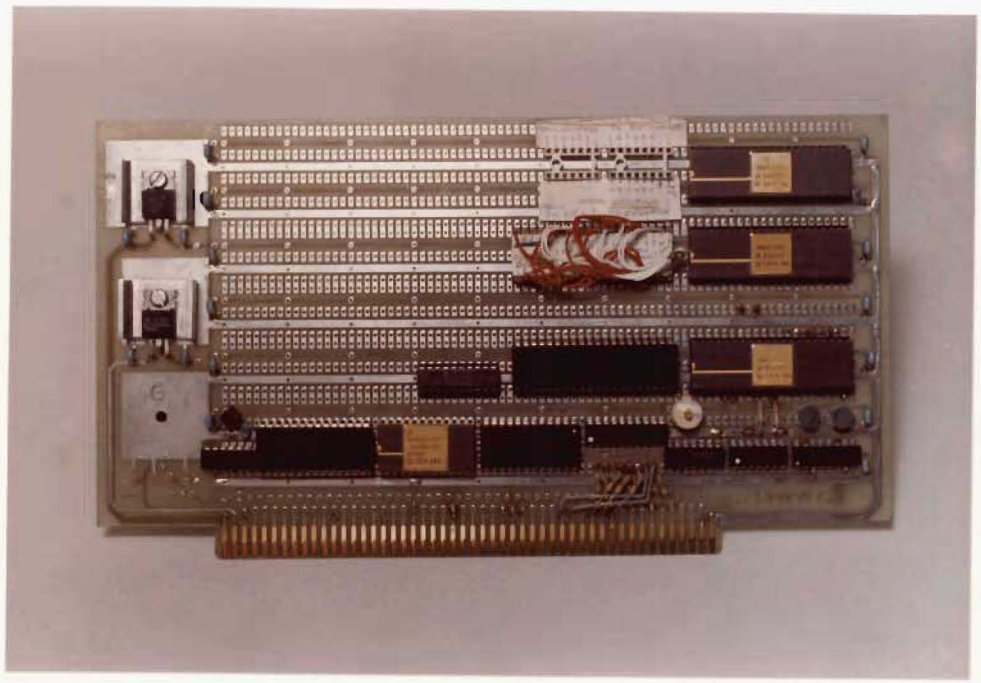
These chips are mounted on a single S-100 printed circuit board shown in Figure 4.8. Also mounted on this board are:

(a) Various ICs providing necessary chip-select and control bus signals.

(b) A small bread board area where the System Timing Controller (STC) signals are routed.

(c) A 16 pin socket which leads the pulse (P1 to P6) and direction (D1 to D6) signals (plus +5V and ground) to the signal conditioning unit.

The signal conditioning unit is shown in Figure 4.9 and performs these tasks:



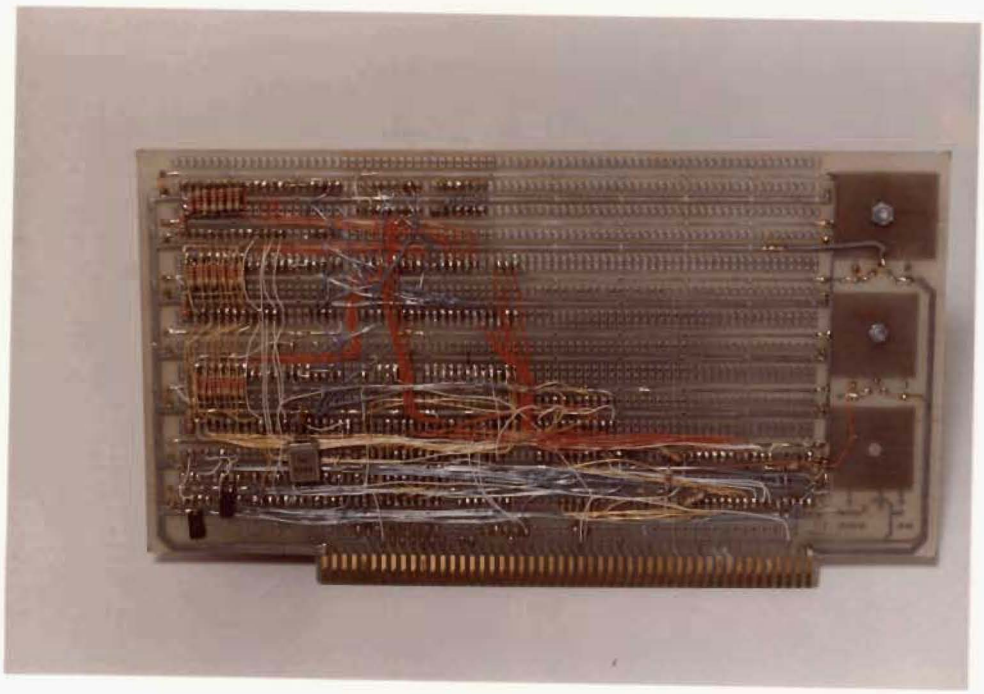


FIGURE 4.8

Top and bottom views of the MCU

(less the signal conditioning unit)

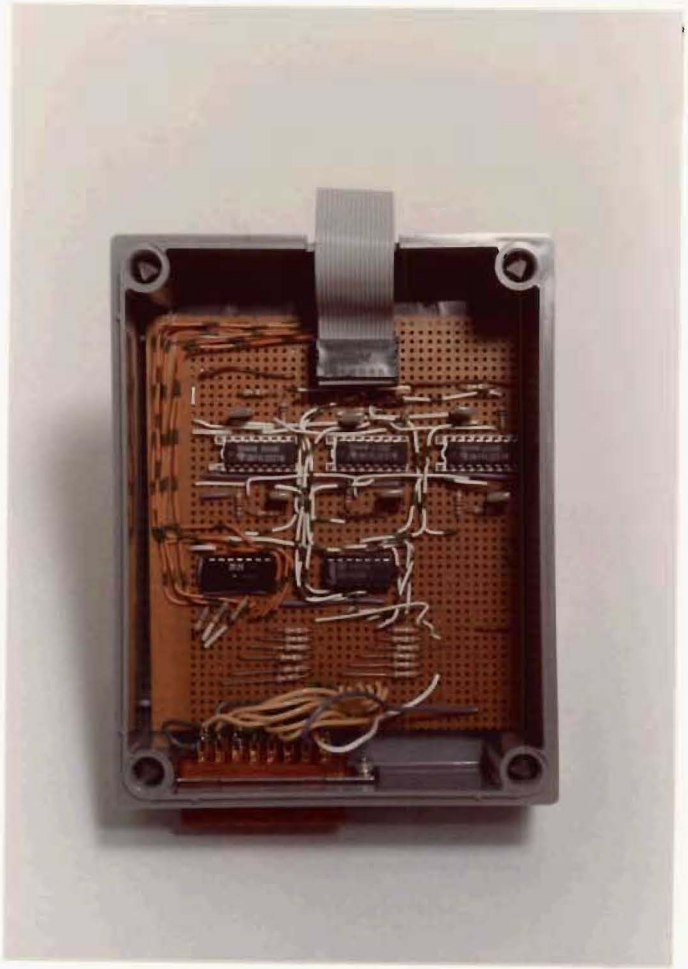


FIGURE 4.9

The Signal Conditioning Unit

(i) The ACU step command pulse on the position line must exceed 50μS duration for detection, so the short pulses emanating from the 6 RSFDGs are each fed through a retriggerable one-shot set at a 75μS period. The chips used are 3x 74LS221s.

(ii) The ACU control input lines are held at 24V and must be pulled low by signals. Thus, all signals must be inverted and buffered from the 24V on the ACU. Thus, all 12 control lines are fed to the ACU via buffers with open collector outputs. Chips used are the 7407 and 7406.

4.3 Theory of Multiple Pulse Stream Control of Stepper Motors

As mentioned in Chapter 3, the movement of the platform can be described by a string of actuator state vectors $\underline{AS}$ and corresponding times.

These vectors must be converted into equivalent data blocks that will cause the MCU to move the actuators to the specified positions in the specified times.

This function is done by the KODE program that receives actuator data and then creates the required MCU data blocks.

The KODE program, given two consecutive actuator vectors $\underline{AS}_{k-1}$ and $\underline{AS}_k$ will produce N data blocks where $N \geq 1$. The reason the movement from $\underline{AS}_{k-1}$ to $\underline{AS}_k$ cannot always be made using one data block is due to limitations in the number of actuator steps that each data block can produce.

This arises because the digital manner of frequency division conflicts with the requirement that the final pulses on each position line all occur at the same time. This is explained below.

4.3.1 The Common Divisor Theory (CD)

Given two consecutive actuator vectors $\underline{AS}_{k-1}$ and $\underline{AS}_k$ we can derive the required movement vector as:

$$\underline{M}_k = |\underline{AS}_k - \underline{AS}_{k-1}|$$

$$= (M_1, M_2, M_3, M_4, M_5, M_6) \qquad (4.3)$$

To achieve these movements we need to create six pulse streams each having a frequency V_i where:

$$V_i = \frac{M_i}{t_k} \qquad (4.4)$$

where t_k is the time allotted for the movement.

If this is done, then the pulse trains will perform as shown in Figure 4.10(a).

Equation 4.4 can be equated to equation 4.2 to give:

$$V_i = \frac{M_i}{t_k} = \frac{F}{SDIV \; . \; RSD_i} \qquad (4.5)$$

Separating values that are common between actuators gives:

$$\frac{Ft_k}{SDIV} = M_i \; RSD_i \qquad (4.6)$$

Therefore we obtain:

$$M_i RSD_i = M_j RSD_j = \frac{Ft_k}{SDIV} \equiv K, \quad 1 \leq i, j \leq 6 \qquad (4.7)$$

The solution for any actuator is:

$$RSD_i = \frac{K}{M_i} \qquad (4.8)$$

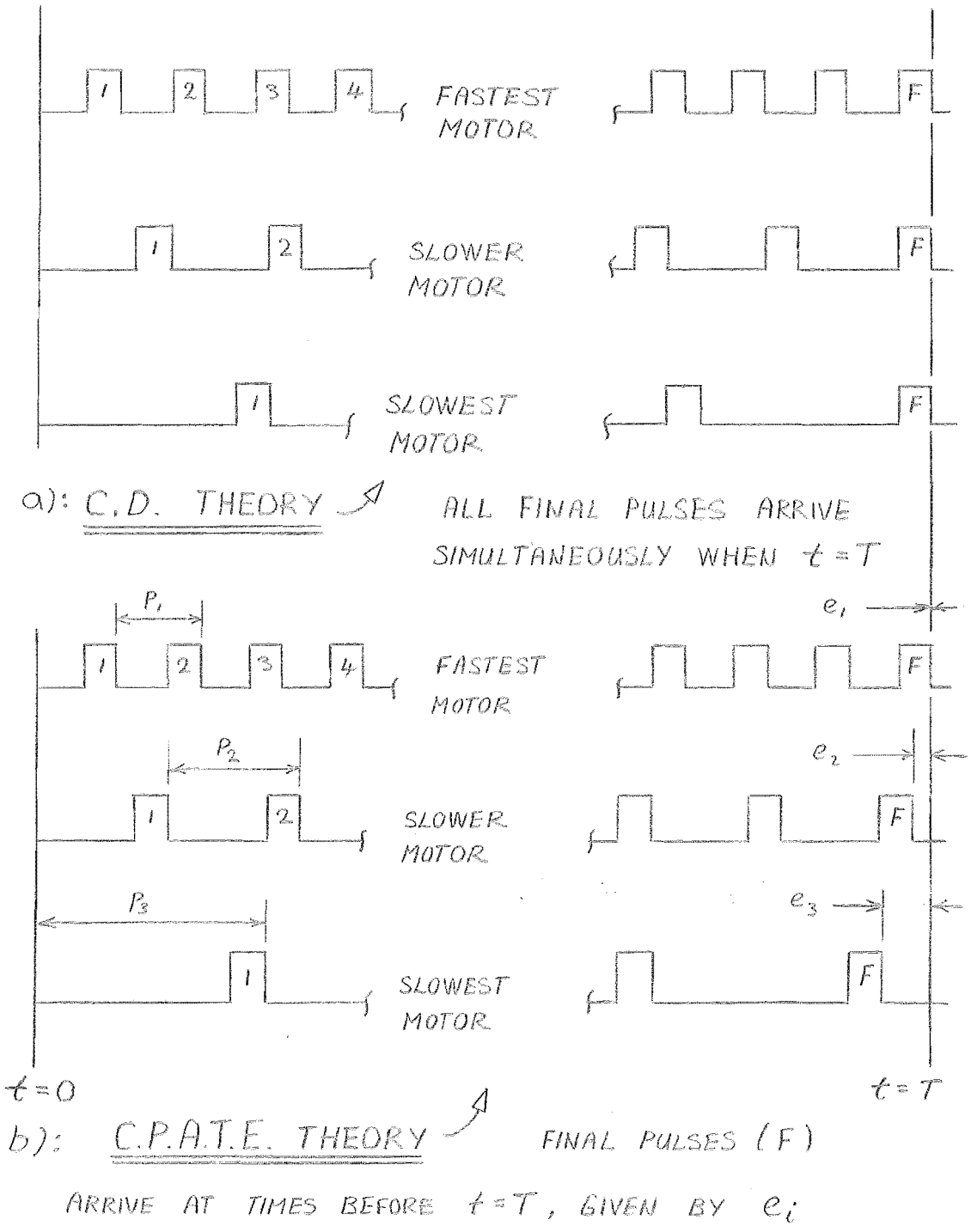


Figure 4.10 a) The C.D. pulse train appearance.

Figure 4.10 b) The C.P.A.T.E. pulse train appearance.

All RSD_i are restricted to integer values however, so M_i must be a factor of K for all i. Thus:

$$K = \frac{Ft_k}{SDIV} = M_1 . M_2 . M_3 . M_4 . M_5 . M_6 \times K_2 \qquad (4.9)$$

where K_2 is any suitable constant.

$$\text{so} \qquad RSD_i = \frac{M_1 M_2 M_3 M_4 M_5 M_6 K_2}{M_i} \qquad (4.10)$$

The required value of SDIV can now be obtained from equation 4.9 giving:

$$SDIV = \frac{Ft_k}{K} = \frac{Ft_k}{M_1 M_2 M_3 M_4 M_5 M_6 K_2} \qquad (4.11)$$

The range of values of SDIV is however restricted to from 20 to 65535. The lower value corresponds to a maximum speed. At this value of SDIV the control over overall speed via SDIV is at a worst value of 5 percent which is considered the worst allowable speed resolution. As SDIV increases, slowing the motors, the speed resolution increases. This affect is unfortunately contrary to what we desire because stepper motors require a greater control of speed at high speeds, particularly in the Slew Range. So the speed resolution of the system has to be "pitched" for the worst case at full speed and is then overspecified at all lower speeds as shown in Figure 4.11.

We can derive an equation relating the maximum speed of the fastest motor to the magnitude of all the movements. It is:

$$\frac{F}{20V_i} = M_2 M_3 M_4 M_5 M_6 K_2 \qquad (4.12)$$

(from equations 4.4 & 4.9)

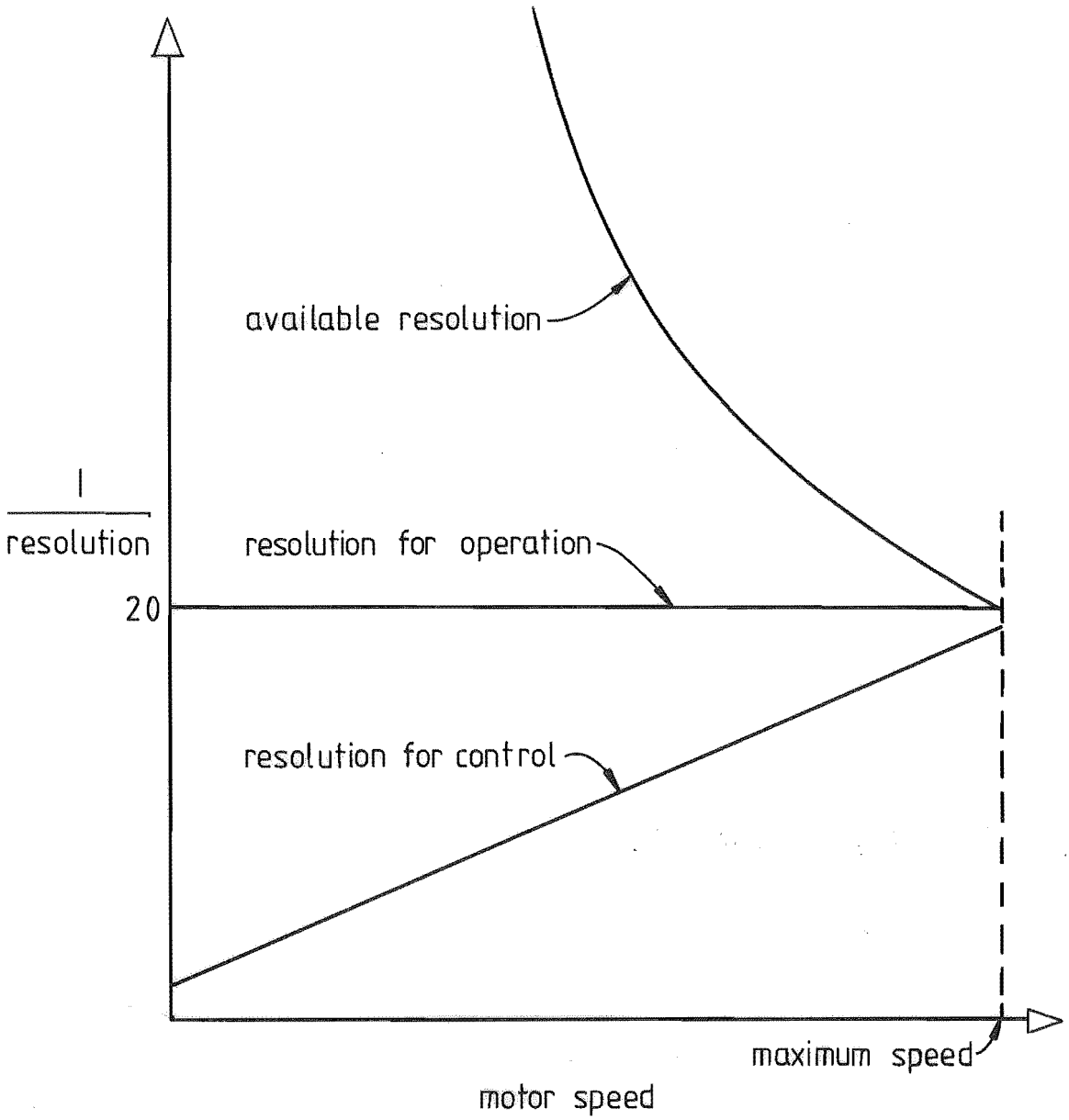


Figure 4.11 Control system resolution characteristics.

where for simplicity, motor 1 is assumed to be the fastest motor. SDIV is put to 20 for maximum speed.

If we assume motors 2 to 6 are to move the same distance (in steps) of M, then we have:

$$M = \left[\frac{F}{20V_1}\right]^{1/5}, \text{ assuming } K_2 = 1 \qquad (4.13)$$

and this gives us an indication of how many steps can be achieved at any maximum speed. This is shown in Table 4.1.

Table 4.1

Movement Capability of CD Theory

Speed of Fastest Motor (steps/sec) V_1	Number of steps per block (steps) M
50	5.8
100	4.8
200	4.2
400	3.6
800	3.1
1600	2.7

This shows that the number of allowable steps in a data block movement is severely limited using the above CD theory. The allowable number increases when fewer actuators are used.

It was concluded that the requirement that the final pulses on all the position lines occur simultaneously was

too restricting and that a small time difference could be allowed in the arrival of the final pulses. The development of this concept is described below.

4.3.2 Controlled Pulse Arrival Time Error Theory (CPATE)

The aim of this theory is to create the desired pulse streams so that the last pulses of the slower motors occur at a fraction of their pulse spacing, denoted by k_i, before the last pulse of the fastest pulse stream. This is shown in Figure 4.10(b).

We assume that the motion of the stepper motors can tolerate the addition of a small increase in step separation as the old block pulse stream is replaced with a new pulse stream at t = T.

Let the time value of this extra pulse separation be denoted by e_i (seconds) for motor i. This value is then normalised by multiplying by the current pulse speed V_i to obtain the "Pulse Arrival Time Error" denoted by k_i:

$$k_i = e_i V_i \qquad (4.14)$$

Ideally k would equal zero, but we are allowing k to exceed zero to increase the capabilities of the control system. The value of k must be kept between 0 and 1 to ensure the correct number of pulses are output.

Whatever speed is programmed to occur after the node, the affect of the pulse arrival time error k is to momentarily (for 1 step) add a negative acceleration to the movement. The maximum allowable value of this, in terms of k could be calculated from theoretical equations of the motor/load combination. This calculation has not

been attempted and the following theory assumes a sufficiently small value of k has been experimentally confirmed.

The relative speed divisor RSD_f for the fastest motor is:

$$RSD_f = INT \left[\frac{Ft_k}{SDIV \cdot M_f} \right] \qquad (4.15)$$

An error will exist in taking the closest lower integer value but this will only be reflected in a small variation in t_k for the movement.

The ideal speed divisors for the other motors are:

$$RSD^I_i = \frac{M_f}{M_i} RSD_f \qquad (4.16)$$

and the actual speed divisors will be:

$$RSD^A_i = INT \left[\frac{M_f}{M_i} RSD_f \right] \qquad (4.17)$$

(Superscripts I and A denote ideal and actual values respectively.)

RSD^A_i will only equal RSD^I_i if there exists a factor of M_i in M_F or RSD_f (as would occur using the CD theory). The maximum difference between RSD^A_i and RSD^I_i is 1, as below:

$$RSD^A_i = RSD^I_i - 1 \qquad (4.18)$$

and the average difference would be ½.

The system is analysed assuming the worst case, so equation 4.18 applies.

The pulse arrival error resulting from equation 4.17 is:

$$k_i = M^A{}_i - M_i \qquad (4.19)$$

where $M^A{}_i$ is the actual number of pulses output.

$$M^A{}_i = \frac{M_f RSD_f}{RSD^I{}_i - 1} \qquad (4.20)$$

(from equations 4.16, 4.17 and 4.18)

and M_i = the ideal number of pulses output assuming a perfect frequency divider.

$$M_i = \frac{M_f RSD_f}{RSD^I{}_i} \qquad \text{from (4.16)}$$

Thus:

$$k_i = \frac{M_f RSD_f}{RSD^I{}_i - 1} - \frac{M_f RSD_f}{RSD^I{}_i}$$

Giving:

$$k_i = \frac{M_f RSD_f}{RSD^I{}_i \ (RSD^I{}_i - 1)} \qquad (4.21)$$

Now,

$$RSD_f = \frac{Ft_k}{M_f SDIV}$$

and

$$RSD^I{}_i = \frac{Ft_k}{M_i SDIV}$$

Giving:

$$k_i = \frac{M_i}{\frac{Ft_k}{M_i SDIV} - 1}$$

$$\Leftrightarrow \ M_i{}^2 + k_i M_i - \frac{k_i Ft_k}{SDIV} = 0$$

V_i can be introduced by using:

$$t_k = \frac{M_i}{V_i}$$

Giving:

$$M_i = k_i \left[\frac{F}{V_i SDIV} - 1 \right] \qquad (4.22)$$

$\frac{F}{V_i SDIV}$ is normally much larger than 1 so we can simplify to:

$$M_i = \frac{k_i F}{V_i SDIV} \qquad (4.23)$$

Substituting into equation 4.23 values of F = 4.9152 MHz, k = 0.05 for a 5 percent maximum pulse arrival error and SDIV = 20 for high speed, gives these movement capabilities per block:

Table 4.2

Movement Capability of CPATE Theory

(With a 5 percent maximum arrival error)

Speed of fastest motor (steps/sec)	Number of steps per block (steps)
50	245
100	122
200	61
400	30
800	15
1600	7

4.3.3 Overall Speed Divisor Relation

The speed divisor (SDIV) is chosen to give a maximum speed of 500 steps/sec at a value of 20. This gives a 5 percent speed resolution at maximum speed. This gives the following relation between the speed of the fastest motor and the speed divisor:

$$V_f = \frac{10,000}{SDIV} \qquad (4.24)$$

This relation, when substituted into equation 4.22 gives:

$$M = \frac{k_i F}{10,000} \approx 24 \text{ steps where } k_i = 5 \text{ percent} \qquad (4.25)$$

This equation imposes a simple upper limit on the number of steps allowable per System Vector.

4.4 Conclusion

Tables 4.1 and 4.2 are graphed in Figure 4.12 which illustrates the superior movement capability of the controlled arrival time theory, for a block movement, when used in a 6 actuator system.

The capabilities of the CD theory however increase and finally surpass the CPATE theory as the number of actuators in the system decreases.

The CPATE theory would appear easier to implement because equation 4.23 imposes a fixed upper limit on M_i for each line at any value of k_i, whereas the CD theory has no such limit, but an overall controlling equation 4.10 which can only be satisfied by a "trial and error" approach.

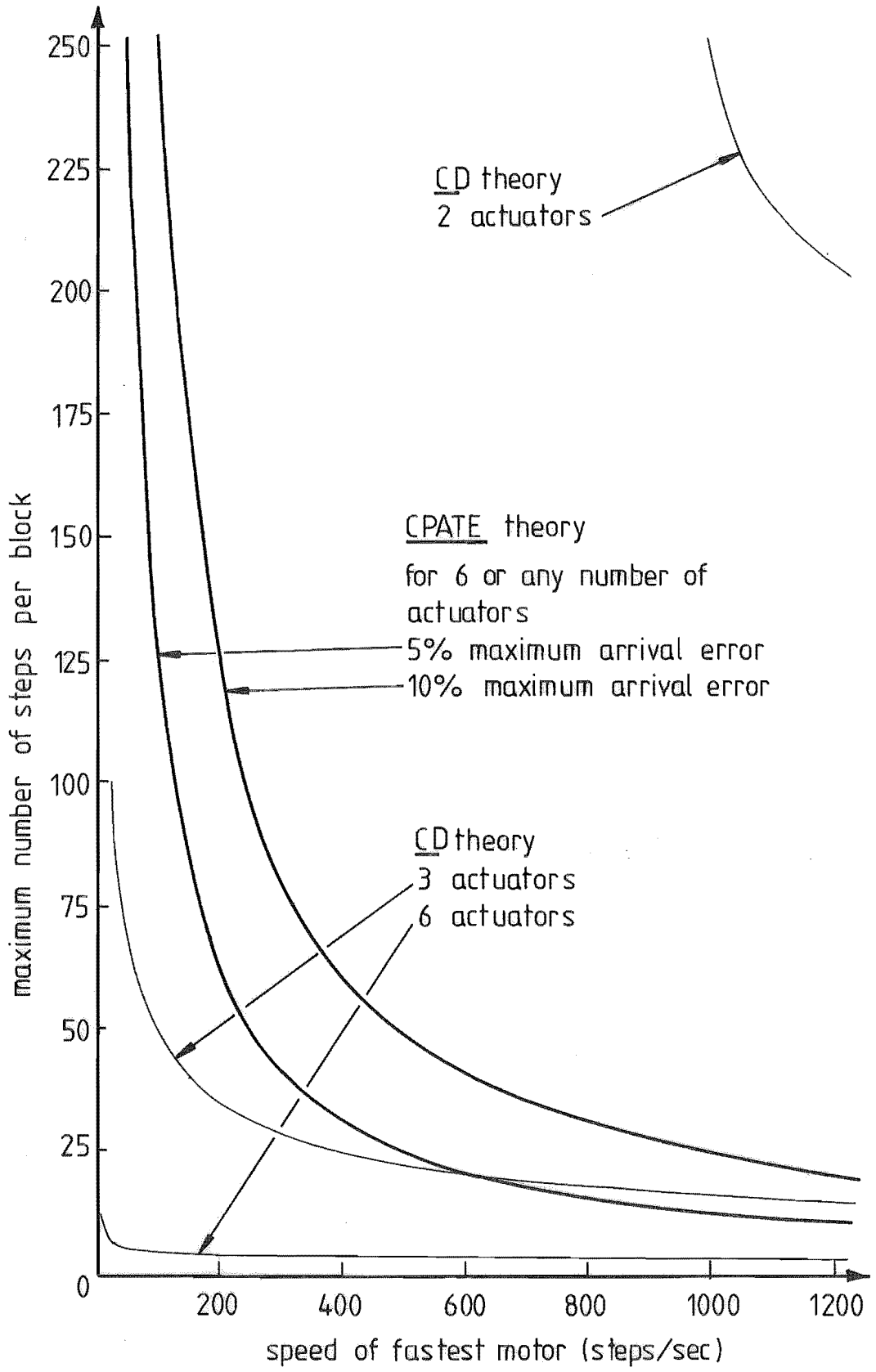


Figure 4.12 Movement capabilities of the C.D. and C.P.A.T.E. pulse train theories.

The CPATE theory was adopted to create the data which generates the block pulse streams. It was chosen in preference to the CD theory because it allows more steps to be made in a block. Another advantage is computational simplicity compared to the CD theory. Also the pulse arrival error is zero on the fastest motor, so any acceleration perturbations due to the CPATE theory's operation do not affect the most sensitive motor.

Even using the CPATE theory however, one data block will normally not be able to generate enough pulses to span the required movement from one Actuator State Vector to the next. This is because the number of steps to move from node to node, even for the smallest resolution, is larger than one data block can generate. (Compare Tables 3.1 and 4.2)

Thus, the movement along a Movement Vector is usually accomplished using many data blocks. These blocks are generated by linearly dividing $\underline{M}_k$ into enough smaller movements so that the number of steps in each movement is within the step capability of each data block. These small movements are called "System Vectors".

Because of the discrete nature of the movements, an exact linear division is not normally possible, so each successive System Vector will vary by up to 1 step per actuator.

CHAPTER 5

THE REQUISITE SOFTWARE

Every aspect of the operation of this Stewart platform is computer controlled or computer based. Thus, the software in itself can be considered the main substance of any description of the operation of the system. So while this chapter seeks to describe the operation of the software, details of operation of the system and of control of the MCU are also illuminated.

Figure 5.1 shows the relations between the computer programs (software), the computer and the various devices connected to the computer.

The software can be split into three parts:

(a) Proprietary software comprising the NEWDOS computer operating system, and the EDIT program.

The NEWDOS system (abbreviated to DOS) is the computer system operating program that creates the initial user interface and controls the calling of disc based programs. Refer to [9] for details.

The EDIT program, called using DOS, is used to create a SOURCE file for a particular desired movement of the platform. Refer to [10] for details.

(b) The COMPILE program (Appendix 2) This program converts the SOURCE file into a corresponding RUN file. The RUN file is used as data by the

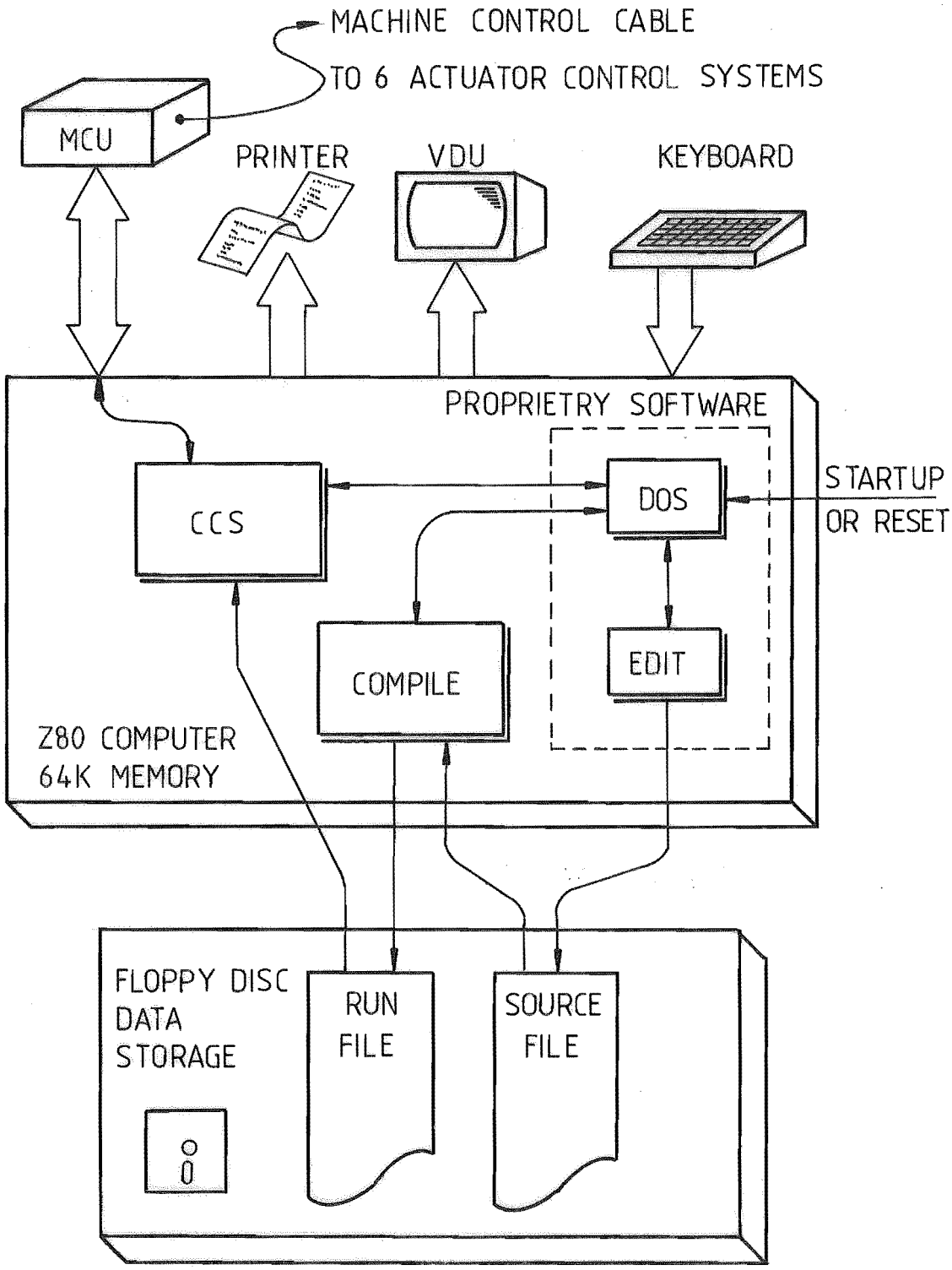


Figure 5.1 The computer - machine control system.

Computer Control System (CCS) to create the desired movement of the platform.

(c) The CCS program (Appendix 3) When this program is called from DOS, the Computer - MCU combination becomes a Computer Control System and is able to call up RUN files and use them to create platform movements. It can also perform other functions such as zeroing the table and moving the table from its present position to the starting position of a movement.

The operations of the COMPILE and CCS programs are now explained in turn.

5.1 The COMPILE Program

The COMPILE program converts the geometric, resolution and speed information in the SOURCE file into data blocks in the RUN file. The overall structure of this program is shown in Figure 5.2.

The reader should refer to Figures 5.3 and 5.4 while reading the following sections. These figures help the reader to understand how a desired movement is produced on the Stewart Platform.

Figure 5.3 shows the overall manipulation of data from a short SOURCE file to create 47 data blocks in a RUN file. The actual movement described by the SOURCE file is shown in Figure 5.4 which shows how the movement is split into many small movements.

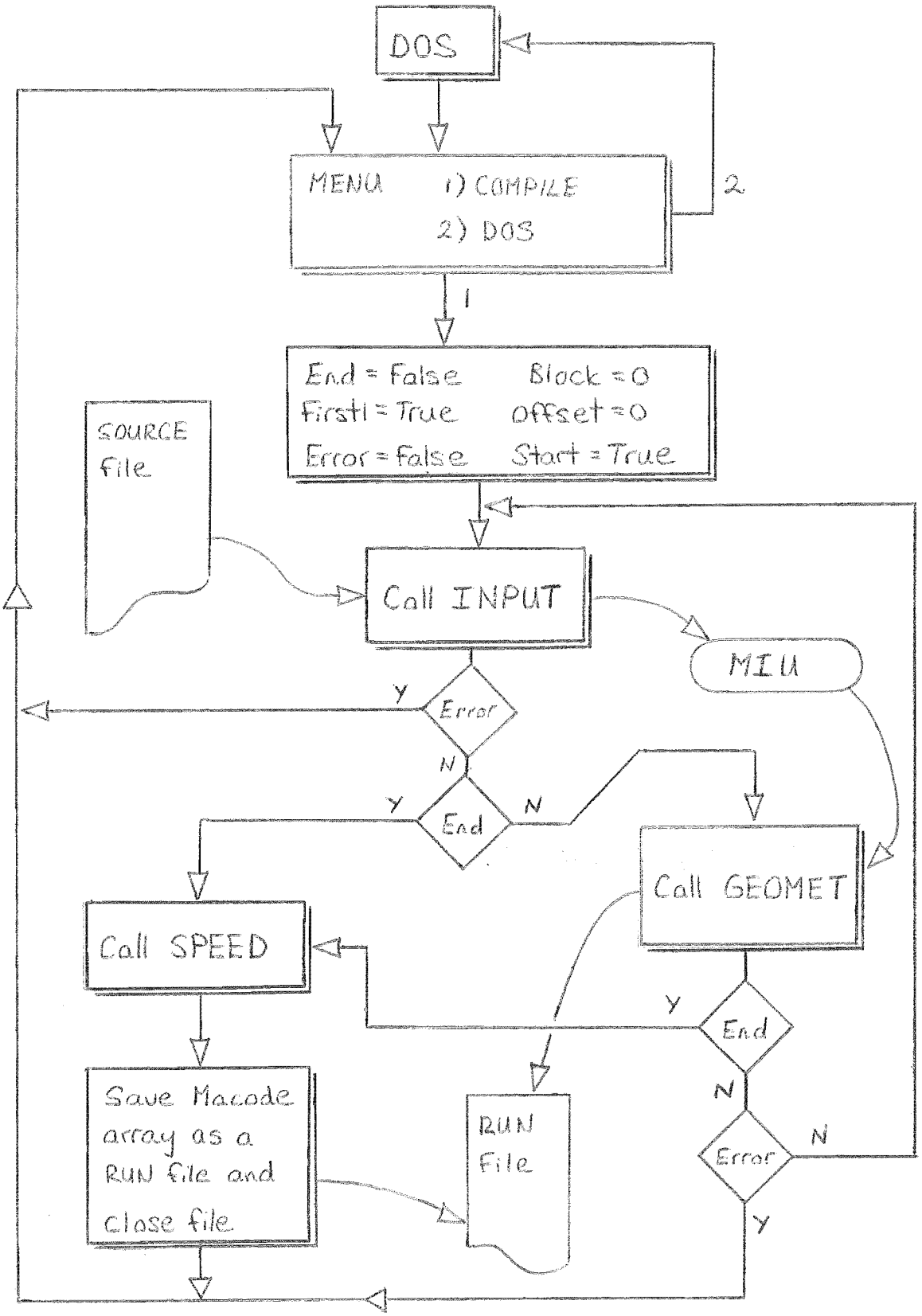


Figure 5.2 The COMPILE program.

The heart of the COMPILE program is the GEOMET subroutine. Contained in GEOMET is the software embodiment of figure 3.3 which describes the overall theoretical method of deriving actuator state vectors $\underline{AS}_1, \ldots, \underline{AS}_N$ given the geometric and path control information as listed below:

$C_S = (X, Y, Z, \phi, \theta, \psi)_S$ -- starting coordinate

$C_F = (X, Y, Z, \phi, \theta, \psi)_F$ -- finishing coordinate

$V = v$ -- path speed

$R = r$ -- path resolution

$\underline{p} = (a, b, c)$ -- platform zero vector

The above information relates to one specific movement command contained on the SOURCE file and will be called a Movement Information Unit, or MIU, as shown in Figure 5.3.

GEOMET takes an MIU and converts the information into a series of actuator vectors $\underline{AS}_1$, to $\underline{AS}_N$. These vectors however cannot be used directly to control the MCU. As explained in section 4.3, each successive actuator pair $\underline{AS}_{k-1}$, $\underline{AS}_k$ is converted, (using the KODE subroutine) into a certain number of data blocks. These data blocks are called MCDB, for "Machine Control Data Blocks". These MCDB, when fed sequentially to the MCU under control of the CCS will produce the desired actuator, and hence platform movement from $\underline{AS}_{k-1}$ to $\underline{AS}_k$.

The MCDB produced by KODE are numbered and placed sequentially into the MACODE array.

The COMPILE program works its way through a SOURCE file feeding GEOMET with successive MIUs from this file.

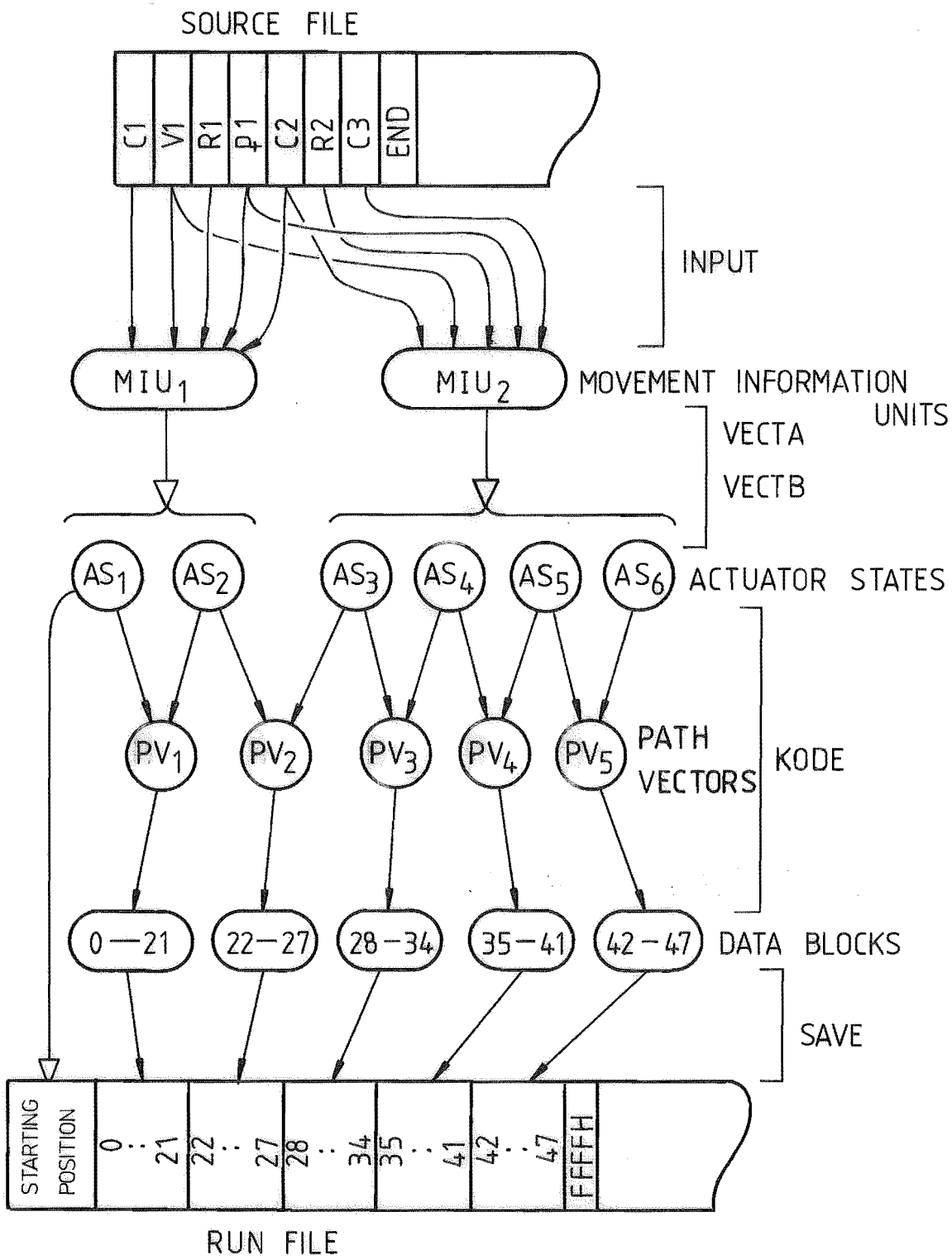


Figure 5.3 Data manipulation of an example Source file.

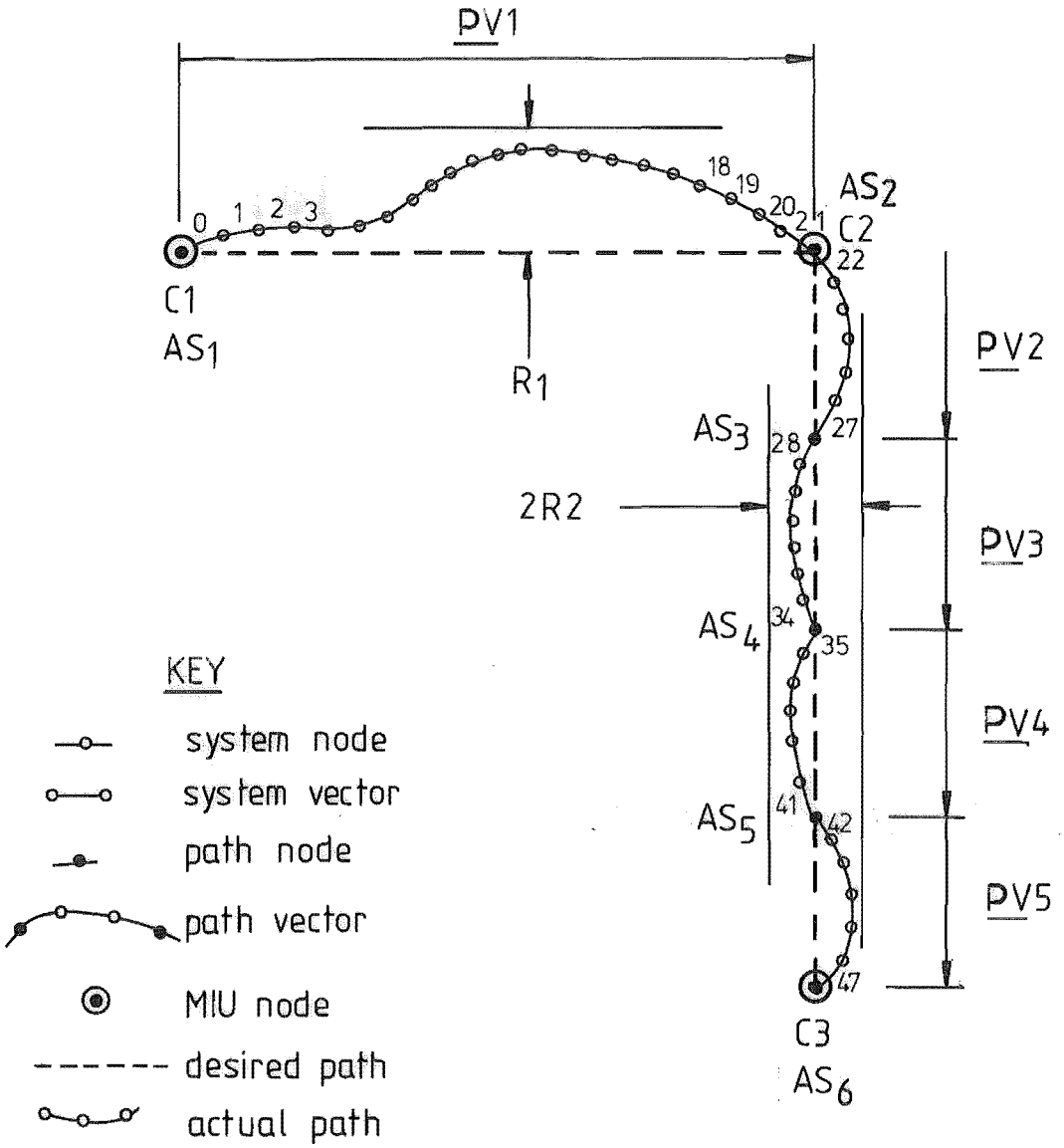


Figure 5.4 Representation of actual movement specified in Source file of figure 5.3.

Each call of GEOMET with a new MIU causes the MACODE array to fill with more sequentially numbered MCDB.

Also, as GEOMET calculates each actuator vector, it saves data pertinent to a subsequent acceleration smoothing program - SPEED into an array called SPD.

The MIUs are fed to GEOMET using the INPUT subroutine which works its way through the source file extracting successive MIUs and also looking for the end of the SOURCE file.

When the end is detected the END flag is set to be TRUE and COMPILE is directed to the left hand branch of Figure 5.2.

All the basic data from the SOURCE file has now been compiled and is resident in the MACODE and SPD arrays. Now SPEED is called. This subroutine uses the data in the SPD array to modify the speed information in many of the MCDB in MACODE. This is done to ensure the smooth and controlled accelerations of the stepper motors when the MACODE data is used to move the actuators. The theory and operation of this aspect of the software is dealt with in Chapter 6.

After the MCDB in MACODE have been speed processed, an "end of file" code is placed at the end of the data in MACODE and this data is saved as a RUN file on disc.

The SOURCE file has been compiled into a RUN file and COMPILE returns to its MENU normally.

COMPILE can return to it's MENU abnormally however. This can be due to either:

(a) An error from LENGTH due to a leg length limit being exceeded,

(b) An error from SPEED due to an excessive acceleration,

(c) An error from INPUT due to incorrect data presentation in the SOURCE file,

(d) An error from KODE if the amount of data exceeds the capacity of the CCS program (1100 MCDB).

5.1.1 SOURCE File Input Format

The SOURCE file must be composed using EDIT to the format shown in Figure 5.5.

Notes to Figure 5.5:

(a) Any sequential line numbering sequence may be used.

(b) Each line must, as shown, have an identifying letter in the first tab stop position. The identification letters have meanings as shown on Figure 5.5.

(c) The relevant data is input immediately after the identifying letter without an intervening blank.

(d) The data is input in standard FORTRAN free format, real number notation. Units used are mm, seconds, and for angular measure, degrees.

FIGURE 5.5

Source File Input Format

```
Line #     tab ---> .
                    .
                    .
XXXX                Ca, b, c, d, e, f      starting coordinate
XXXX  Starting      Vv                     starting speed
XXXX  MIU           Rr                     starting resolution
XXXX                Pg, h, i               starting platform zero
                                           vector
XXXX                Ca, b, c, d, e, f      second coordinate
XXXX                [Vv]                   optional speed
                                           modification
XXXX  Subsequent    [Rr]                   optional resolution
      MIUs                                 modification
XXXX                [Pg, h, i]             optional zero vector
                                           modification
XXXX                Ca, b, c, d, e, f      following coordinate
XXXX                 END
```

a, b, c: X, Y, Z coordinates of tip of zero vector (mm)

d, e, f: Euler angles giving orientation of tip of zero vector (degrees)

g, h, i: position of tip of zero vector on the platform (mm)

v: path speed (mm/sec)

r: path resolution (mm)

[ ]: optional data

Thus, a new coordinate C in six space where:

C = (100.0, 200.0, -50.0, 10.0, 30.0, -90.0)

would be input as:

xxxx TAB-->|C100.,200.,-50.,10.,30.,-90.

And a path resolution of 0.2 mm would be input as:

xxxx TAB-->|R0.2

(e) The first five lines of information must be input, and in the order shown. These lines represent the first movement (MIU) of the file. Then, from 0 to 50 MIUs, as shown, may be entered. These represent any subsequent desired movements.

(f) Note that every MIU must have a coordinate value, but that inclusion of new path control information is optional. If new path control information isn't specified, previously defined values apply. The information must be input in the order shown.

5.1.2 The INPUT Program

The flow diagram for the INPUT program is shown in Figure 5.6.

The program is directed down the left hand branch on the first call of INPUT. First the SOURCE file is loaded into memory from disc. Then the first five lines of the file are translated into the first MIU. Any error of omission or mismatch of identifying letters will cause an abnormal return to the MENU. Otherwise INPUT returns to

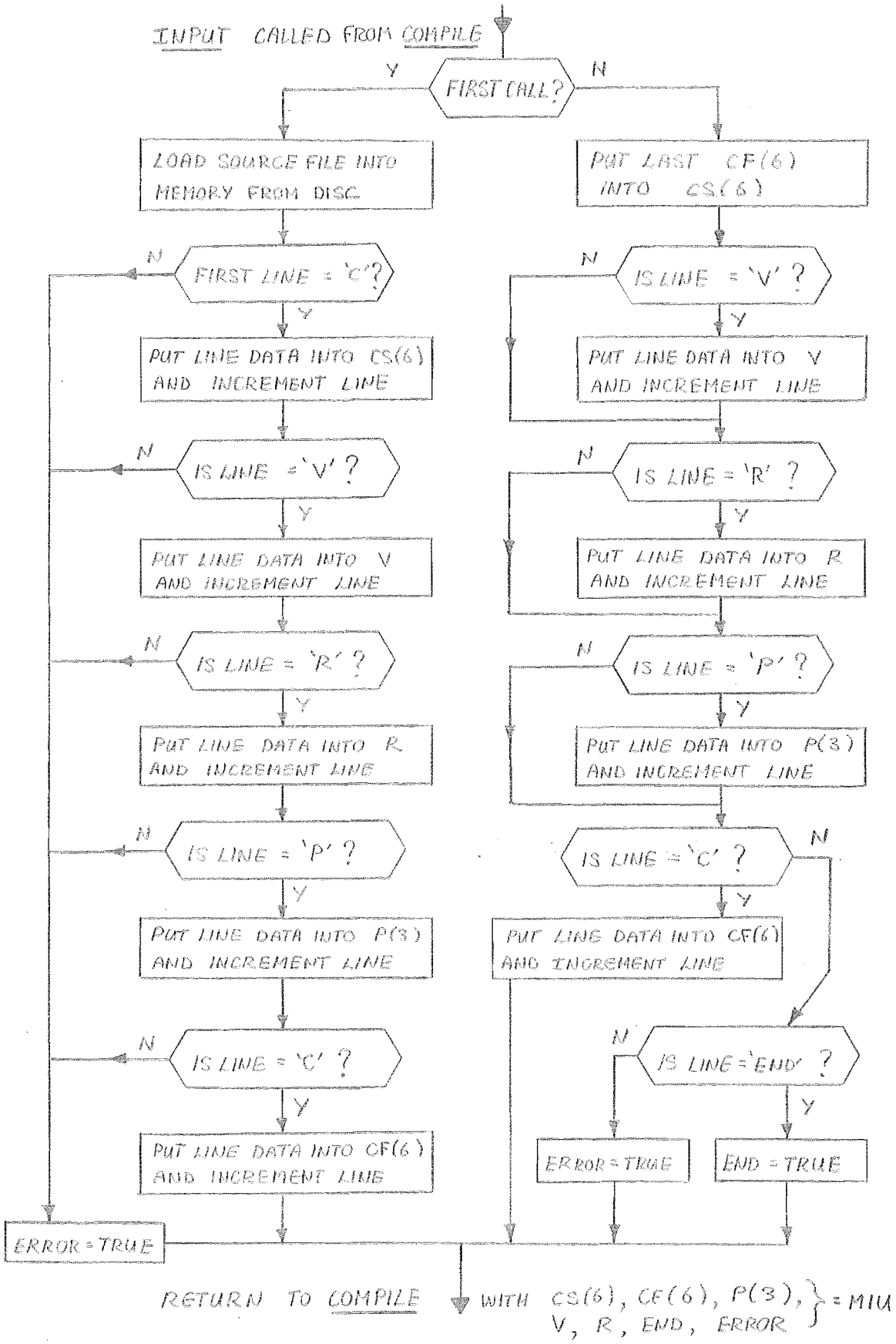


Figure 5.6 The INPUT program.

COMPILE with the first MIU. After this MIU has been compiled by GEOMET, INPUT will be called a second time. Hereafter, the right hand path will be followed.

The right hand path first puts the C_S of the new MIU as the C_F of the preceding (or first) MIU. Then the V, R and $\underline{p}$ values of the last MIU are updated optionally from the SOURCE file lines. Then, a "C" line is expected. If a "C" line is found C_F is updated with the data on that line and INPUT returns to COMPILE with a new MIU, which will be compiled by GEOMET, and INPUT will be called again to supply a new MIU from the right hand path.

Thus, compilation of the SOURCE file will continue until a "C" line is not detected in the next SOURCE line. If there is no "C" an "END" line is expected. If this is found INPUT will return to COMPILE with an END = TRUE flag which will initiate the terminal phase of the COMPILE program. If there is no "C" and an "END" line is not found, then INPUT returns to the MENU abnormally.

5.1.3 The GEOMET Program

The overall operation of GEOMET is shown in Figure 5.7.

First, initial data is calculated in VECTA (Figure 5.8). VECTA looks at a single movement from C_S to C_F given the path control information V, R and $\underline{p}$ (all from the input MIU). This movement is called a Control Vector.

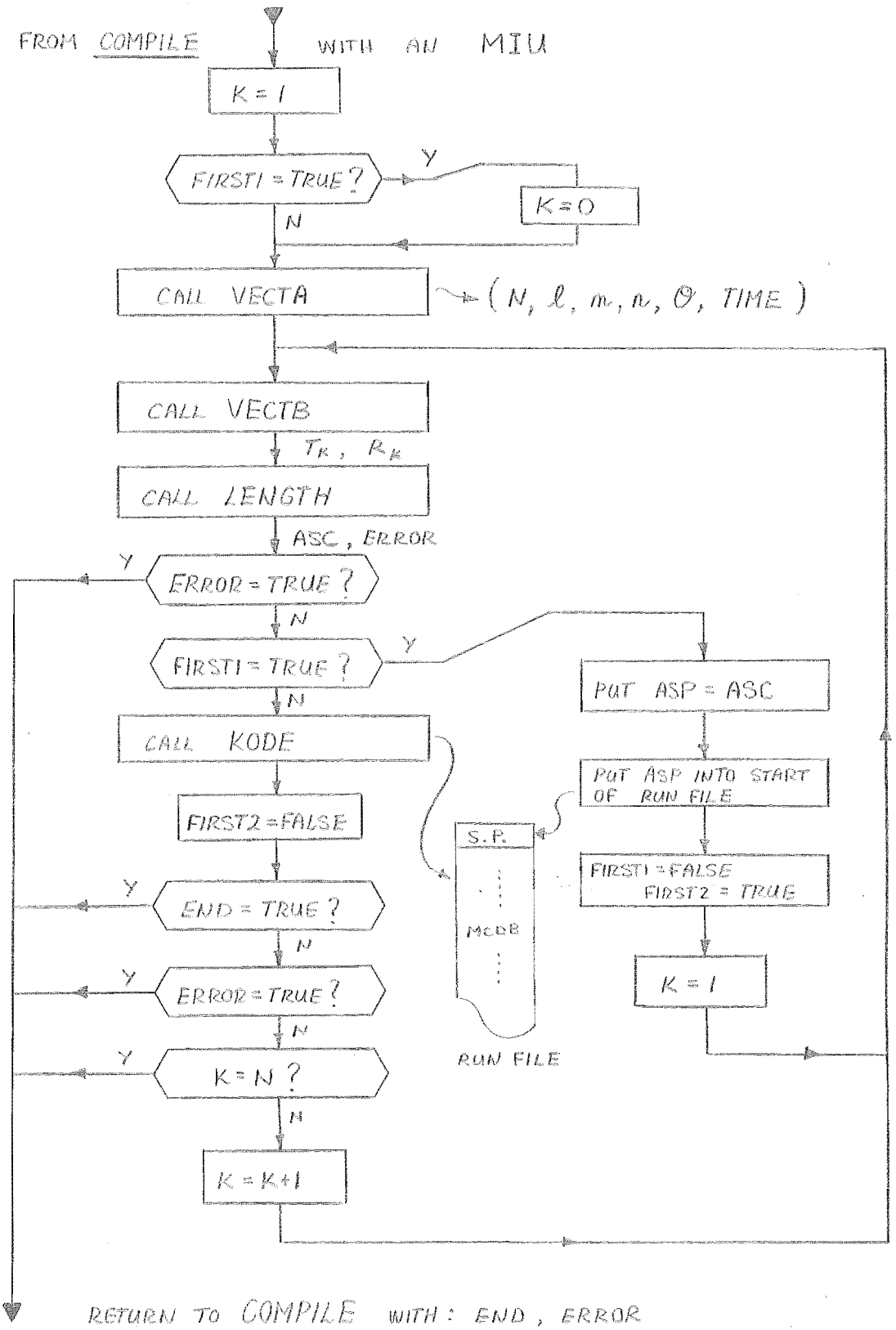


Figure 5.7 The GEOMET program.

VECTA then calculates:

(a) The rotational matrix operator parameters l, m, n and α, and the translational vector $\underline{T}$ that will achieve a single movement along the Control Vector.

(b) N, the number of equal movements this single movement must be divided into to achieve the required resolution of basic path control. These movements are called Path Vectors.

(c) TIME, the number of seconds required for movement along a Path Vector (equal for all path vectors spanning a control vector).

Now the program goes into a loop involving VECTB, LENGTH and KODE which is executed N times. Every loop calculates successive actuator state vectors $\underline{AS}$ using VECTB and LENGTH. KODE converts each new $\underline{AS}_k$, and the previous ($\underline{AS}_{k-1}$) into enough data blocks to achieve the movement (Path Vector) from $\underline{AS}_{k-1}$ to $\underline{AS}_k$.

Each $\underline{AS}$ is calculated using these constant values:

l, m, n - direction cosines of line about which rotation (if any) is taking place in the Control Vector

α - the gross angular rotation of the Control Vector

$\underline{T}$ - the gross translation of the Control Vector

and this variable value:

k/N - where k is the loop counter and N is the fixed loop count target.

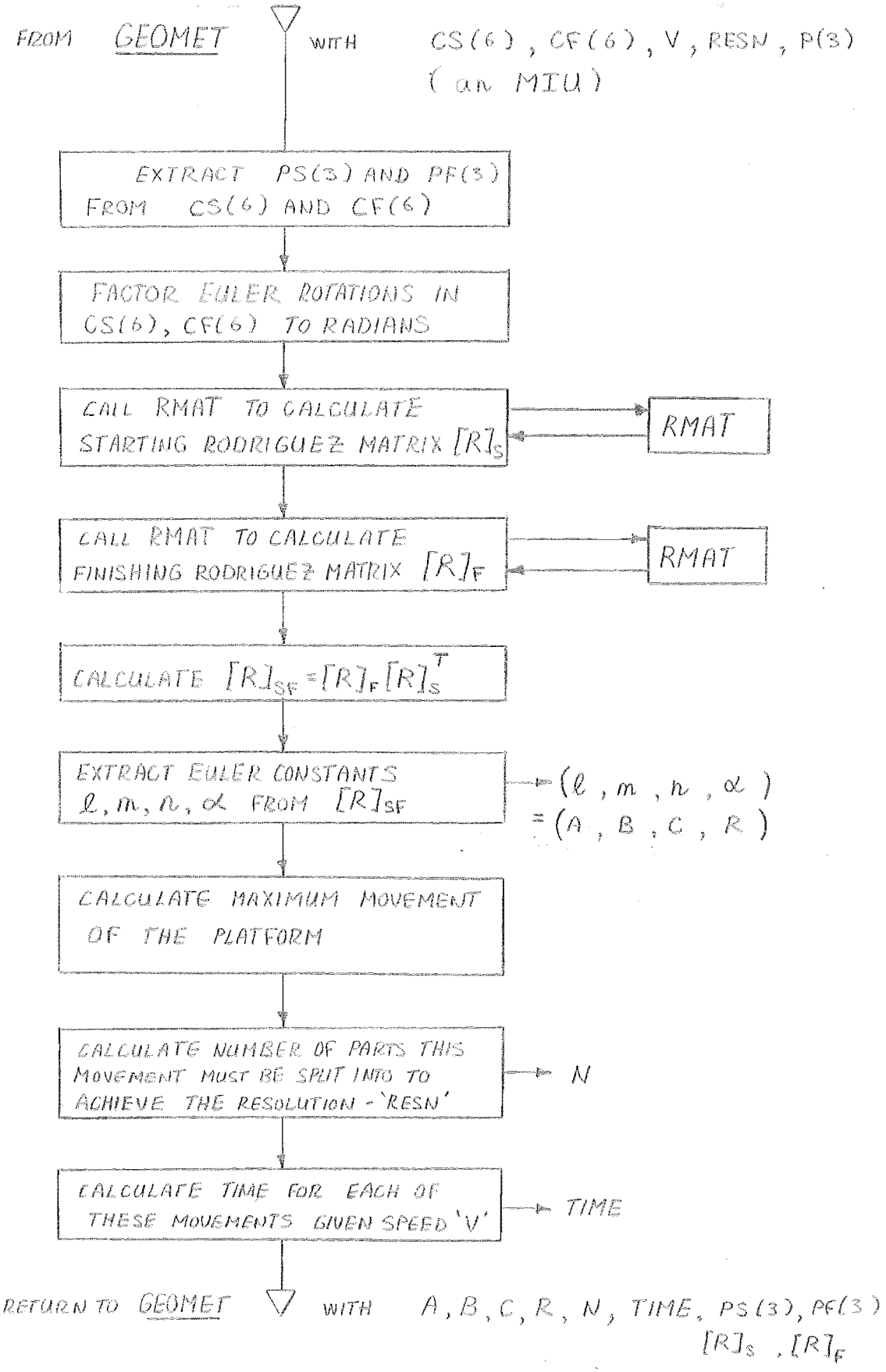


Figure 5.8 The VECTA program.

k/N is used to divide the control vector into N geometrically equal increments.

The calculation of each $\underline{AS}_k$ is a two stage process.

The first stage is done by VECTB (Figure 5.9). It is a purely general operation determining the translation vector $\underline{T}_k$ and the rotational operator matrix $[R]_k$ for the k'th position of the platform as it moves along the Control Vector.

The second stage uses LENGTH (Figure 5.10) to convert this general geometric information $\underline{T}_k$ and $[R]_k$ into the actuator vector, $\underline{AS}_k$ which is specific to this Stewart platform. LENGTH gets dimensional information for the platform kinematics from the data subprogram PHYSIC.

Then, this $\underline{AS}_k$ and the last actuator state vector $\underline{AS}_{k-1}$, plus other data is sent to KODE (Figure 5.11). KODE calculates the movement vector in leg space (steps) that will change the actuator states from $\underline{AS}_{k-1}$ to $\underline{AS}_k$. This vector is called the Path Vector.

KODE will probably have to split this Path Vector into many smaller vectors because of MCU operational limitations. These vectors are called System Vectors and correspond to the MCDB used to control the MCU. KODE then translates the System Vectors which have units of steps and time into equivalent MCDB which can control the MCU to create the desired System Vector movement on the actuators.

This loop in GEOMET is executed with ascending values of k until it is finally executed with k = N. When this is done the control vector will have been spanned with N Path

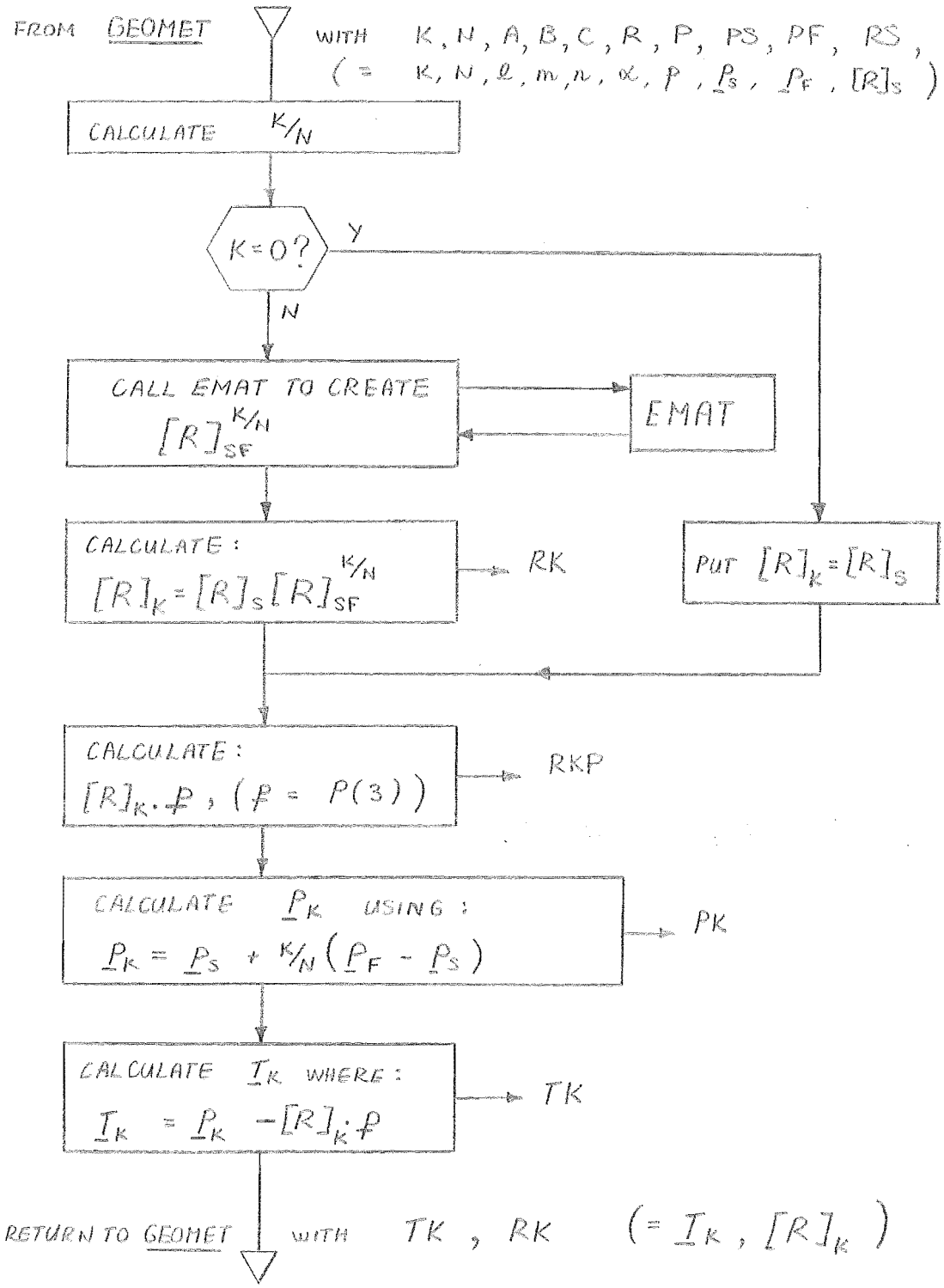


Figure 5.9 The VECTB program.

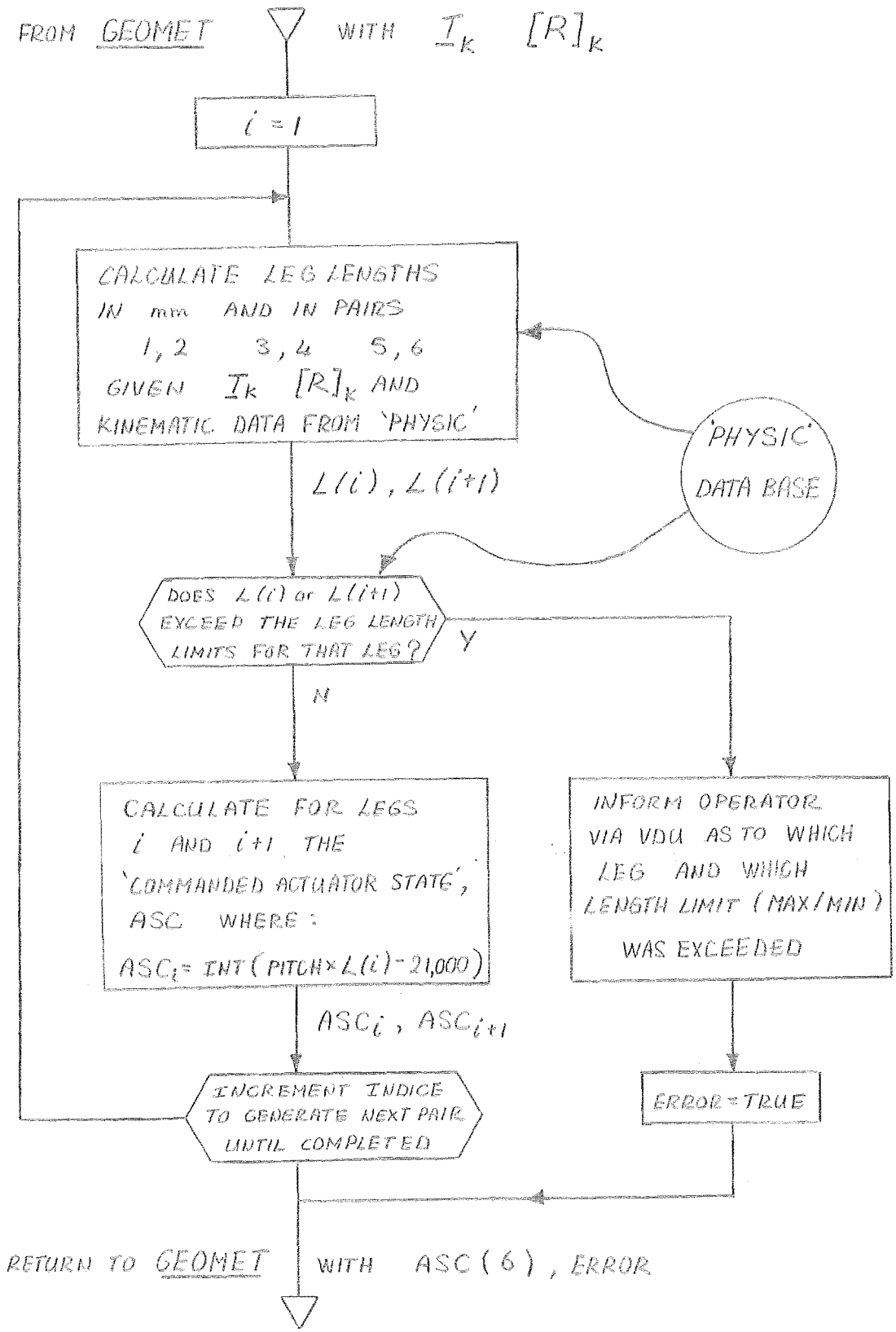


Figure 5.10 The LENGTH program.

Vectors, with each Path Vector being spanned by many equal System Vectors.

Each System Vector will have been converted into an equivalent MCDB and stored in the MACODE array by KODE.

GEOMET will then return to COMPILE having converted the MIU into many MCDB.

There can be several modifications to the normal execution of this loop as described below:

(a) On the first call of GEOMET there is no last $\underline{AS}$ vector available. So, the first vector, $\underline{AS}_0$, is calculated normally and is placed in the appropriate variable, but KODE is not called as no Path Vector can be calculated.

The starting actuator state vector $\underline{AS}_0$ is also placed at the beginning of the RUN file for use by the CCS.

FIRST is made FALSE and the loop is restarted and will execute normally with an initial k = 1.

(b) An error message generated in LENGTH will cause a program abort back to the MENU.

(c) KODE may detect that the number of data blocks in the MACODE array will overflow the size of MACODE.

If this is detected, the present MCDB is MACODE will be processed by "SPEED" using data from the SPD array and then saved in the RUN file on disc. Once this is done normal processing of Path Vectors can continue. The new MCDB and speed data though will start filling the MACODE and SPD

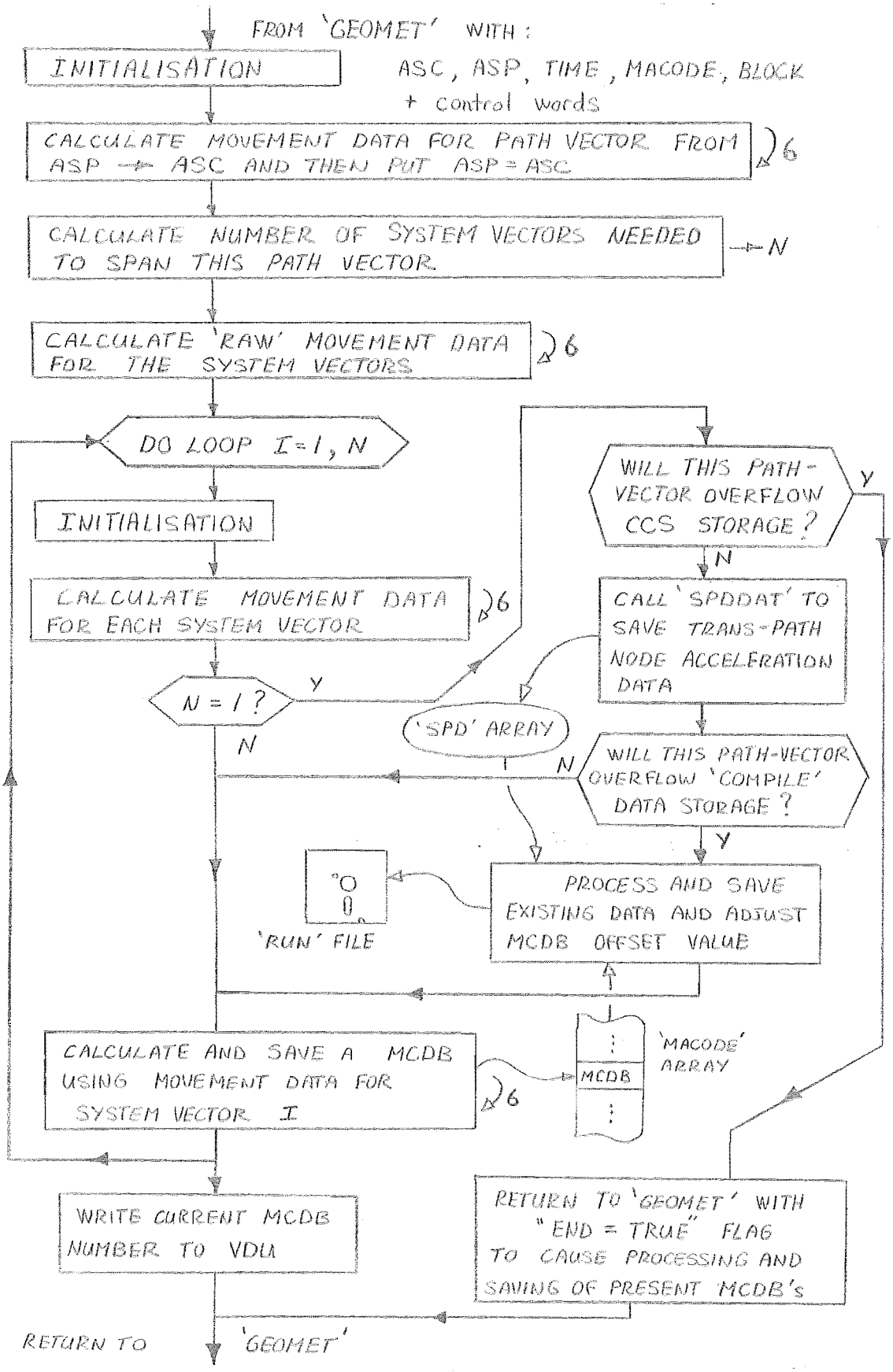


Figure 5.11 The KODE program.

arrays from the bottom, overwriting old data.

This "multi-stage" processing and saving of data was found necessary because the MACODE array size was restricted to 450 MCDB whereas the corresponding array in the CCS can accept 1100 MCDB.

(d) KODE may detect that the processing of a Path Vector will lead to a total number of MCDB that exceeds the storage capacity of the CCS program (= 1100 MCDB).

If this occurs, KODE will return to GEOMET with a premature END = TRUE flag which will cause the speed processing and saving of the present MCDB by the left hand branch of GEOMET. COMPILE will then return to the MENU, having only partly compiled the SOURCE file.

If no fatal errors occur, then GEOMET will continue to process MIUs from the INPUT program until INPUT returns to GEOMET with an END = TRUE flag. This will cause a normal ending sequence, down the left hand branch of Figure 5.7.

GEOMET will then return to the COMPILE menu normally and the SOURCE file will have been "compiled" into a corresponding RUN file, ready for use by the CCS program.

5.2 The Computer Control System Program (CCS)

This program turns the Computer-MCU system into a control system that controls the movements of the Stewart platform. The operations of the CCS are shown in Figure 5.12.

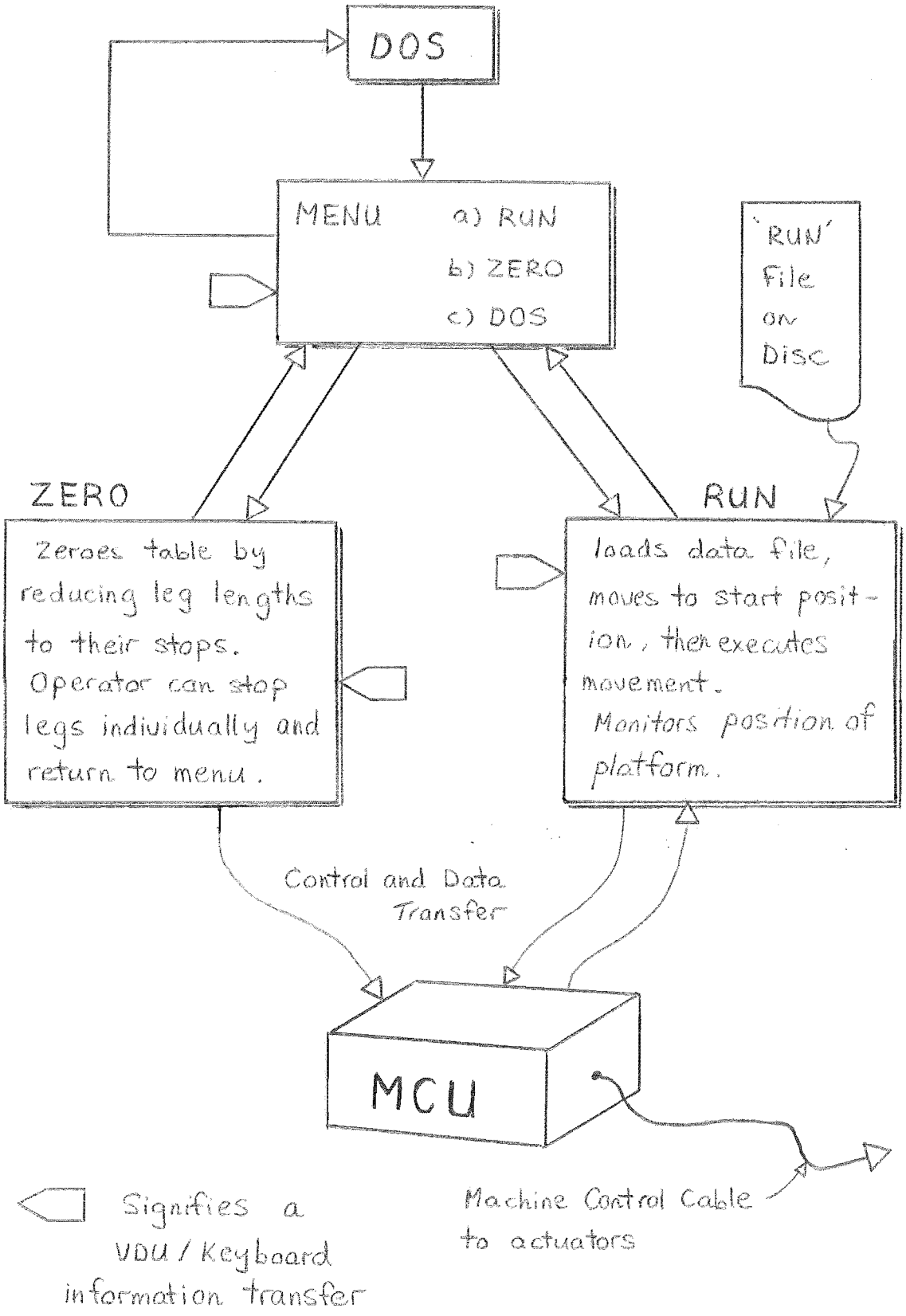


Figure 5.12 The CCS program.

5.2.1 The ZERO program

ZERO (Figure 5.13) is called by the operator from the menu. ZERO is used to place the table into a state where the values of all the actuator positions are known. This position is where all the legs have been moved to their "zero stop" positions. Once this has been done, the internal array giving the present states of all actuators - ASP, can be given the values of the actuators in their zero positions - ASZ.

ASZ has been determined to an accuracy of +/- 25 steps using leg length measurements when the legs are against their zero stops.

ZERO moves all the legs onto their zero stops by calling the GO program with a data block that causes a continuous decrease of each leg length at a speed that is input by the operator.

The operator serves as a visual feedback device. Any leg's downward movement can be stopped when the operator sees that that leg has reached its zero stop. To enable this, the keyboard is scanned for keys 1 to 6 and the BREAK key.

Pressing any of keys 1 to 6 causes a DISARM signal to be sent to the RSFDG controlling that motor's movement. This stops the movement of that leg.

Pressing BREAK causes the OSFDG to be stopped (stopping all motors), initialises ASP to ASZ and returns the operator to the menu.

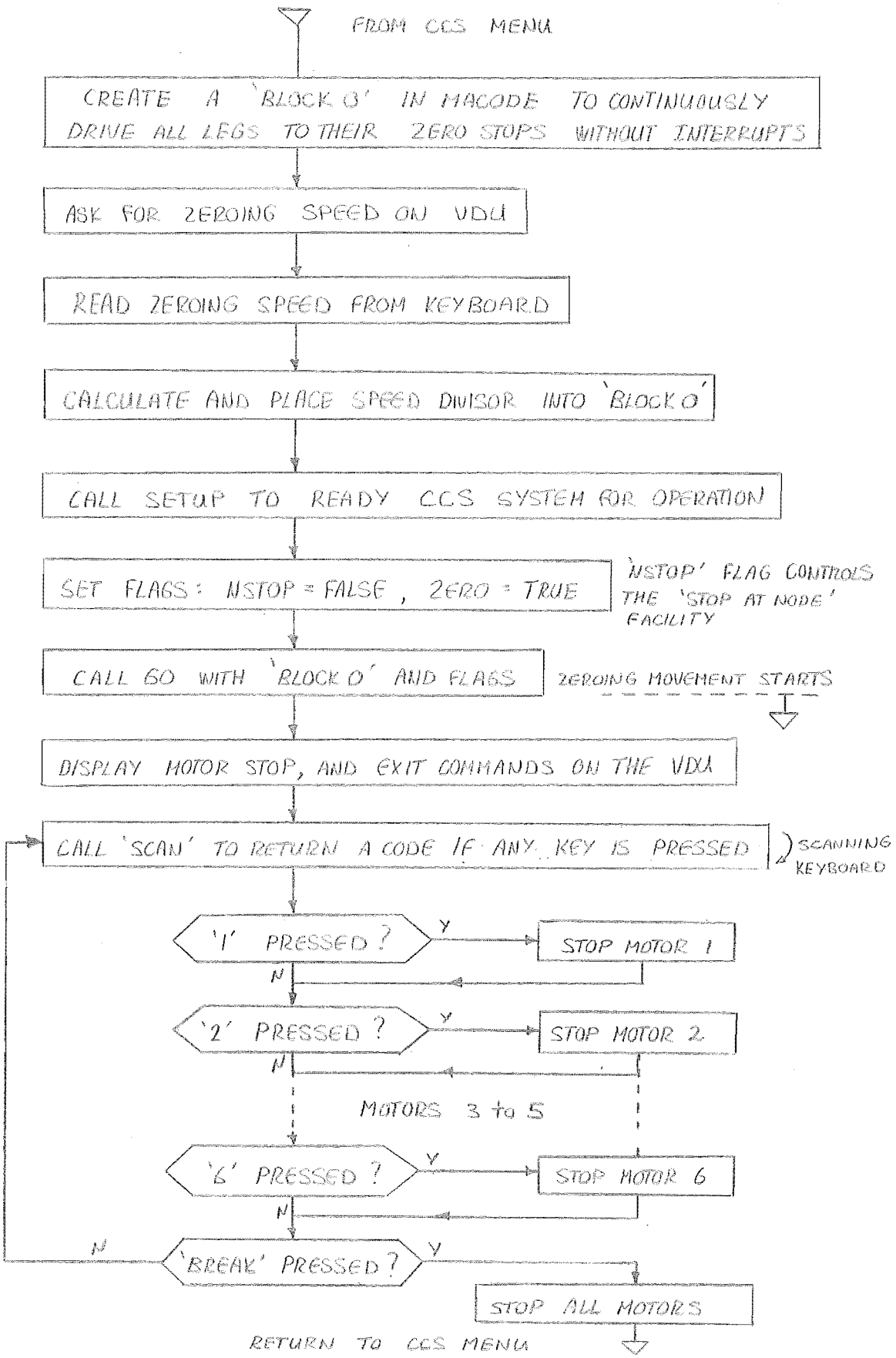


Figure 5.13 TheZERO program.

5.2.2 The RUN Program

The RUN program performs these functions:

(a) RUN, using data from a RUN file controls the MCU to cause the platform to execute the motions originally described by the SOURCE file.

(b) RUN updates the actuator state array (ASP) of the platform after every System Node has been passed.

(c) RUN interfaces with the operator via the VDU and keyboard so the operator can control the operations taking place.

The RUN program operation is described by Figure 5.14.

The execution of the movements in a RUN file is actually comprised of two separate parts described below.

Firstly, the platform must be moved from its present position (described by ASP) to its commanded starting position (described by ASSC). This is done by the top part of Figure 5.14 labelled "STARTING MOVEMENT".

Its present position (in actuator space) may be ASZ because the table has just been zeroed, or it's present position may be the last position of a previous movement. In any case, its present actuator state is normally available in ASP.

The ASP is not available however if the CCS program has just been called up from disc. Thus, the platform must always be zeroed on startup so the CCS knows where movements start from - ASZ.

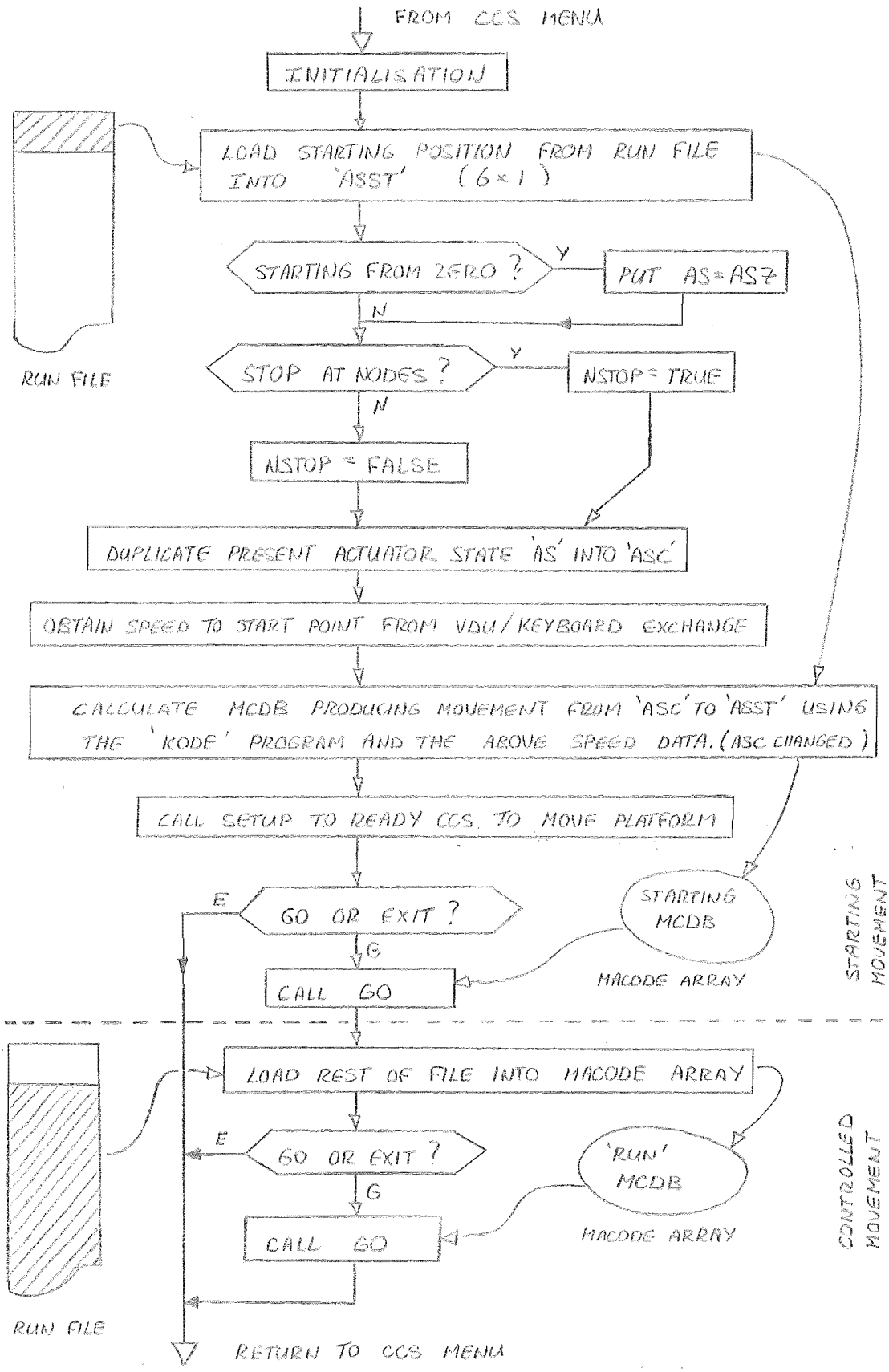


Figure 5.14 The RUN program.

The ASSC is contained in the first 12 bytes of the RUN file. Thus, sufficient information is available to enable a movement from ASP to ASSC. The operator is asked to supply a speed.

Then, a slightly simplified form of KODE is used to calculate the System Vectors and hence MCDB necessary to span the path from ASP to ASSC. The data produced is placed in a large array (11,000 integers) that is subsequently used to store the RUN file MCDB. Double use of this array saves much storage space.

The movement from ASP to ASSC is a point-to-point movement with no path control save speed of the fastest motor.

After all the required MCDB have been calculated, GO is called to execute the starting movement to ASSC. Once this movement is executed the first part of RUN is complete and now ASP = ASSC.

Now, the desired movement of the platform, originally contained in the SOURCE file can be executed using the MCDB's in the RUN file. This is done by the bottom part of Figure 5.14 labelled "Controlled Movement".

The MCDBs of the RUN file are loaded into the MACODE array from disc. This array has space for 1,100 MCDBs.

Then, GO is called to execute the movement. Movement can be stopped, restarted by pressing the "space" bar or the "enter" key respectively.

Completion of the movement causes a return to the MENU and ASP will now have a value corresponding to the finishing state of the last movement.

5.2.3 The GO Program

As described in the preceding section, GO is called to start execution of a movement. The operations of GO are shown in Figure 5.15.

Basically, GO initialises the control system with:

(a) "Mode of operation" data to the MCU which uses the highly programmable GPIC and STC integrated circuits.

(b) With initial data for the first movement (System Vector) of the platform.

Then, it starts motion by arming all the STC counter groups, the OSFDG last.

It then goes into a scanning mode where it just scans for signals that indicate the movement should be stopped. These signals can be:

(i) An operator controlled stop if the "Space" bar is hit. This causes the OSFDG to be disarmed which stops motion. Then GO scans for "Enter" which restarts motion by arming the OSFDG, or "Break" which returns the operator to the CCS menu.

(ii) A "return" flag is found to be TRUE which indicates an error or end of movement. Motion is stopped and the program returns to the CCS menu.

Thus GO only initiates the motion.

The actual control of motion is done by the MCUISR program which calls RELOAD and NODE. The operation of these programs is now explained.

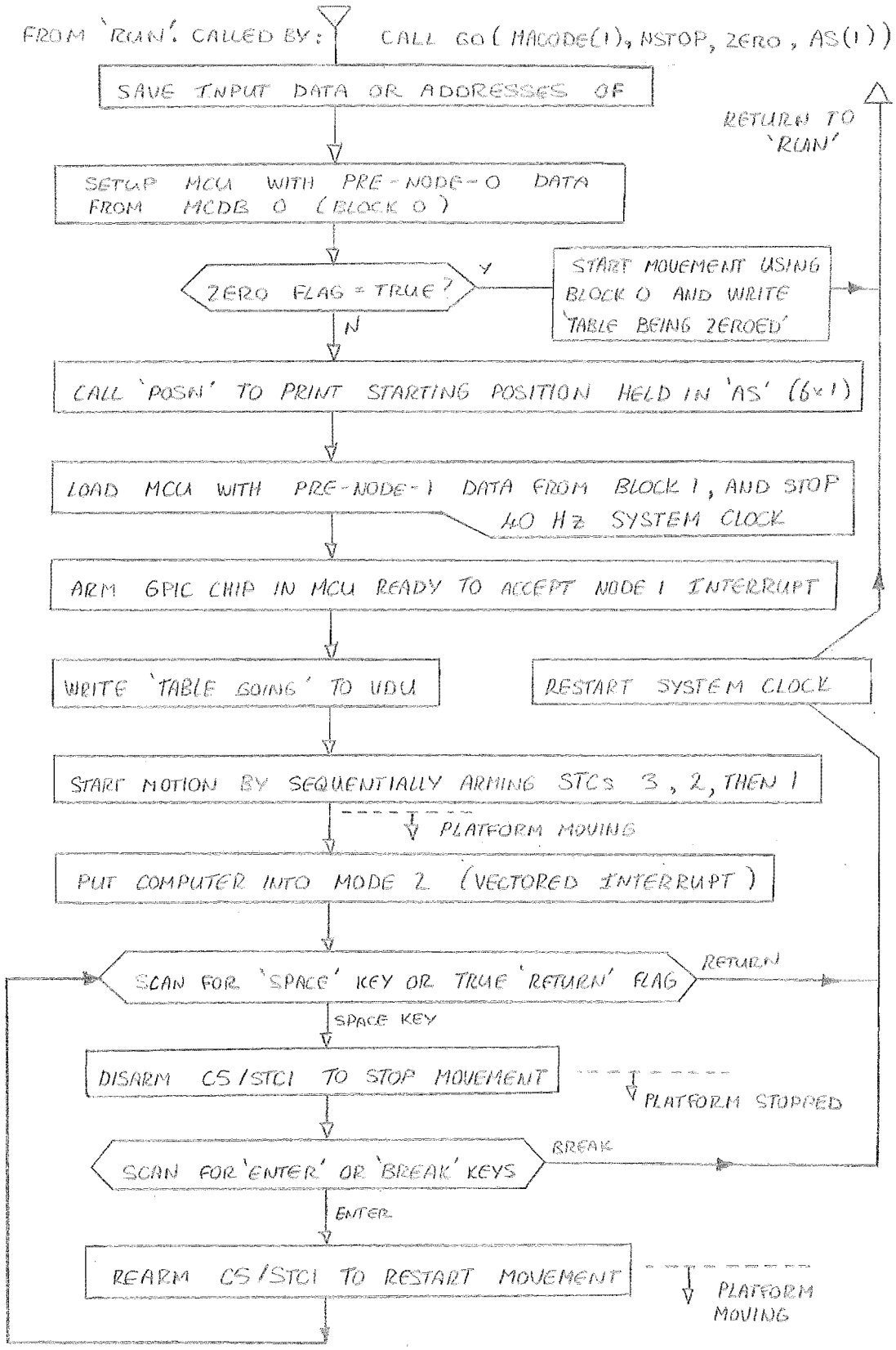


Figure 5.15 The GO program.

5.2.4 The MCUISR Program

This program is the main controlling function of platform movement. It is called by a vectored interrupt from the MCU whenever a System Node is reached.

The operations of the MCUISR program are shown in Figure 5.16. These are sequentially:

(a) Interrupts are disabled so the MCUISR program cannot be interrupted.

(b) Pulse generation and hence movement is stopped by disarming the OSFDG and positional data is saved into "SAVE" registers on the STC chips ready for later retrieval and use by NODE.

(c) A new direction control word DIRG is output and "latched" (memorised) to the MCU on an 8-bit flip-flop. DIRG creates new direction control logic levels on the 6 direction control wires of the machine control cable.

DIRG also controls, via its bit 7, the reload source of the OSFDG and RSFDGs 1 to 6.

The reload source, using bit 7 of DIRG is made to alternate between the LOAD register and the HOLD register as each successive System Node is passed.

This is done so that all the essential motion data can be placed on the STC chips in the LOAD or HOLD registers before each System Node is reached. This allows the data for the following System Vector to be "activated" by just a few commands from the computer, rather than having to

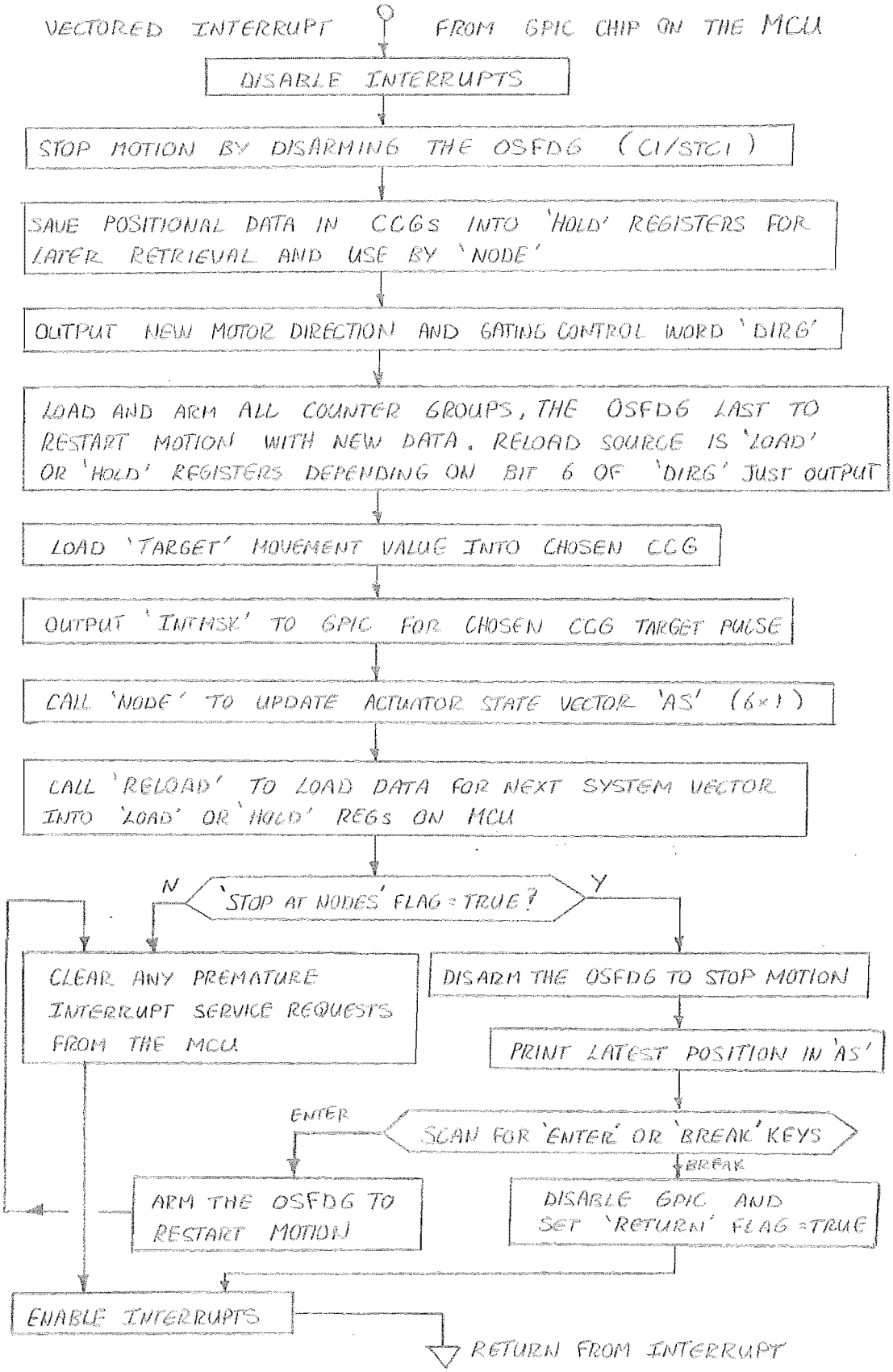


Figure 5.16 The MCUISR interrupt service routine.

load all the data from the computer as it is required.

This greatly reduces the "dead" time between successive System Vectors when the MCU is reconfigured with new motion data.

The duration of this "dead" period spans about 100 CPU clock states which corresponds to 100 µS as the CPU operates on a 1 MHz clock. During this time a motor rotating at 500 steps/sec will have rotated 5 percent of 1 step.

(d) The new motion data is "activated" by the "Load and Arm all Counters" command which is sent to the STC chips 3, 2 then 1 consecutively. The loading (or "reloading") source is either the LOAD or HOLD register depending on the logic level of bit 7 of DIRG which has just been output and "latched" to the MCU. This level is fed to a "special gate" on all the STC chip counter groups. This gate is programmed to control the reload source of each counter depending on the logic level on the gate pin. See Figures 4.6 (a) and (b).

The OSFDG is located on STC chip 1, so the last "LOAD and ARM all Counters" command to STC1 should restart the movement.

(e) It was found that occasionally the OSFDG would not start producing pulses when the "LOAD and ARM" command was issued to it. Thus a second ARM command is sent to the OSFDG to ensure this.

After the motors have started on their new System Vector movements MCUISR executes other tasks that must be accomplished before the next System Node is reached. These are:

(f) The target step count "TARGET" for the fastest motor is loaded into the CCG of that motor, so that the end of the movement (System Node) can be detected.

(g) A mask word "INTMSK" is sent to the GPIC so that the GPIC only "looks" at the output of the CCG of the fastest motor. This is because the other 5 CCGs can generate false "end of movement" pulses from old TARGET data in their target registers.

(h) NODE is called. NODE uses the data which was saved in operation (b), and the last direction word, LSTDRG to update the present actuator state vector, ASP.

(i) RELOAD is called to load the speed divisor and direction divisor data onto the STC chips ready for the next System Node. It loads to the LOAD or HOLD registers depending on the logic value of bit 7 of DIRG for the coming System Vector.

RELOAD also extracts INTMSK, DIRG, TARGET, PORT and REG from the current data block and places them in memory locations ready for use or to be loaded to the MCU in operations (b), (e) and (f) of the next call of MCUISR.

(j) MCUISR checks for a "Stop at Node" command which enables the operator to step through a movement.

(k) The GPIC is cleared of any pending interrupts. This is done before the correct interrupt to erase any premature interrupts that may have been passed to the GPIC before the correct INTMSK and TARGET data were loaded.

(l) Finally interrupts are re-enabled allowing an MCU "end of movement" vectored interrupt signal to be serviced by MCUISR once again.

5.2.5 The RELOAD Program

This program, shown in Figure 5.17 extracts MCDB data from the MACODE array at the bidding of the MCUISR program.

It always increments its data pointer after each data access, so working its way along the array (by 20 bytes per call).

It checks the file block number with its internal block counter first on every call, and if correct proceeds to feed the MCU with data. If the block numbers do not agree it assumes the last block has been passed and initiates the termination of movement by activating the LSTINT interrupt routine (Figure 5.18)

5.2.6 The NODE Program

This program, shown in Figure 5.19 extracts step count data from the "SAVE" registers of the STC chips after every system node has been passed. It either adds or

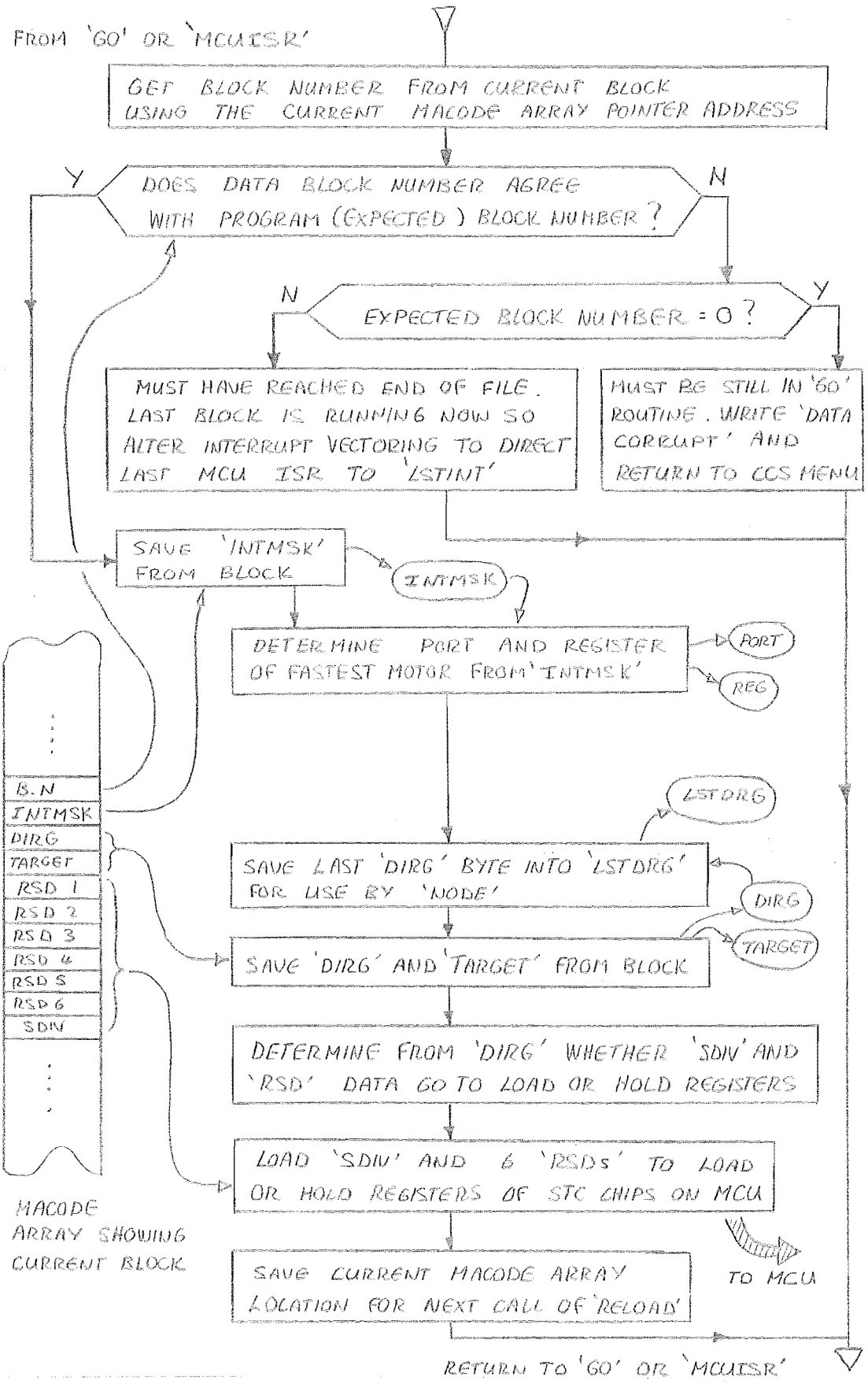


Figure 5.17 The RELOAD program.

subtracts this data (depending on the associated movement directions of the last System Vector) to the ASP array by calling the UPDATE program (Figure 5.20).

Thus the ASP array is updated with the current actuator state vector after every System Node has been passed.

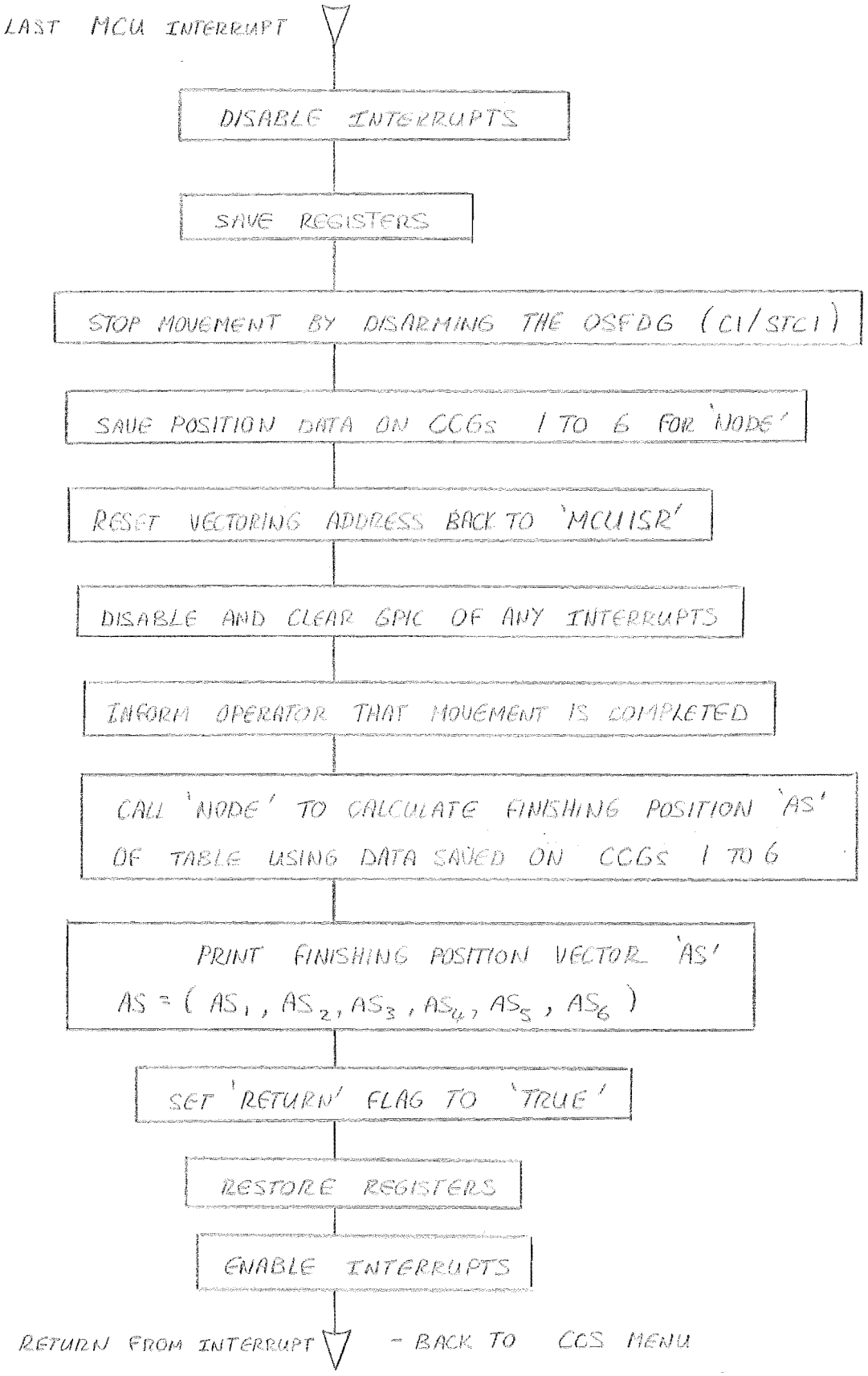


Figure 5.18 The LSTINT interrupt service routine.

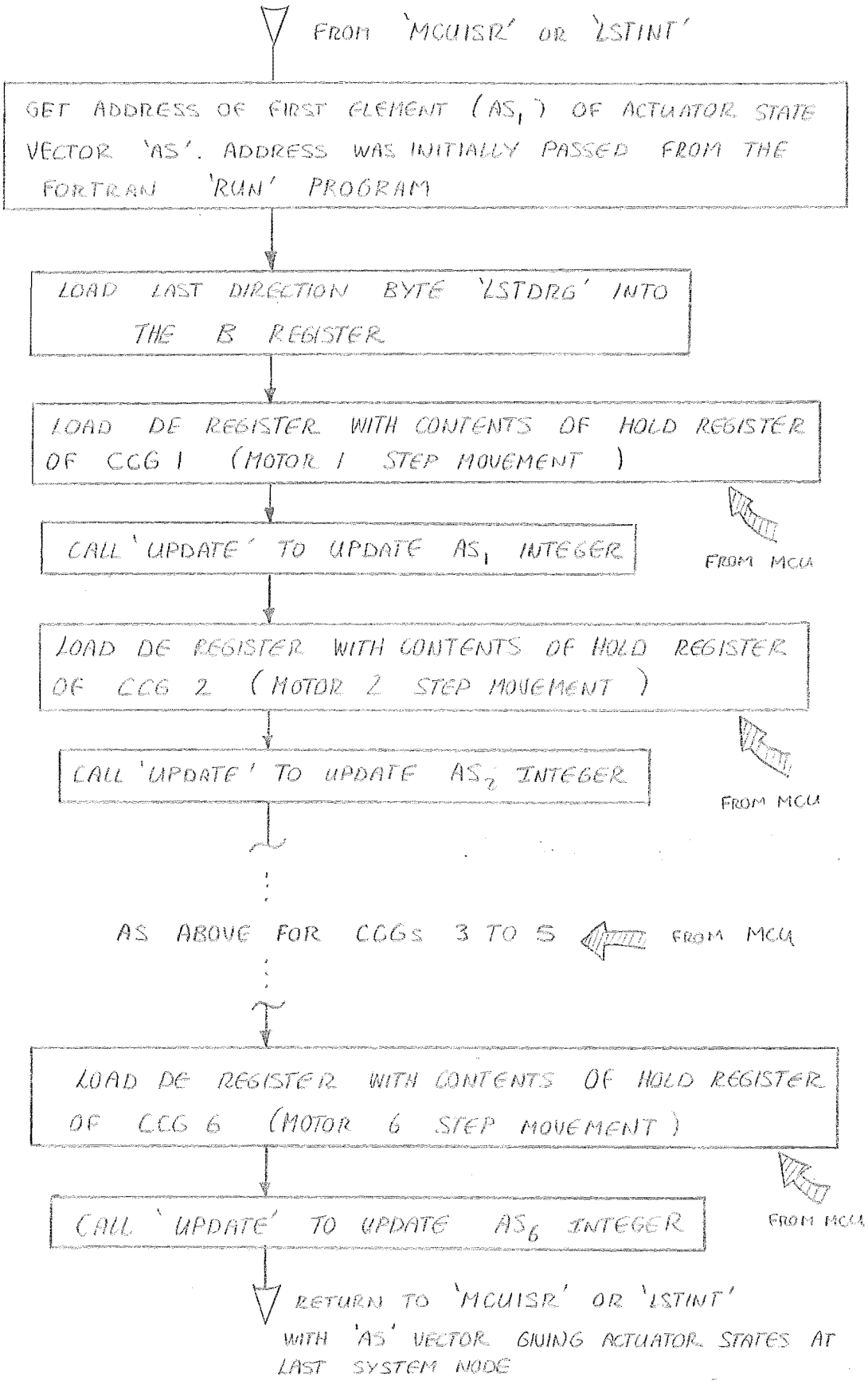


Figure 5.19 The NODE program.

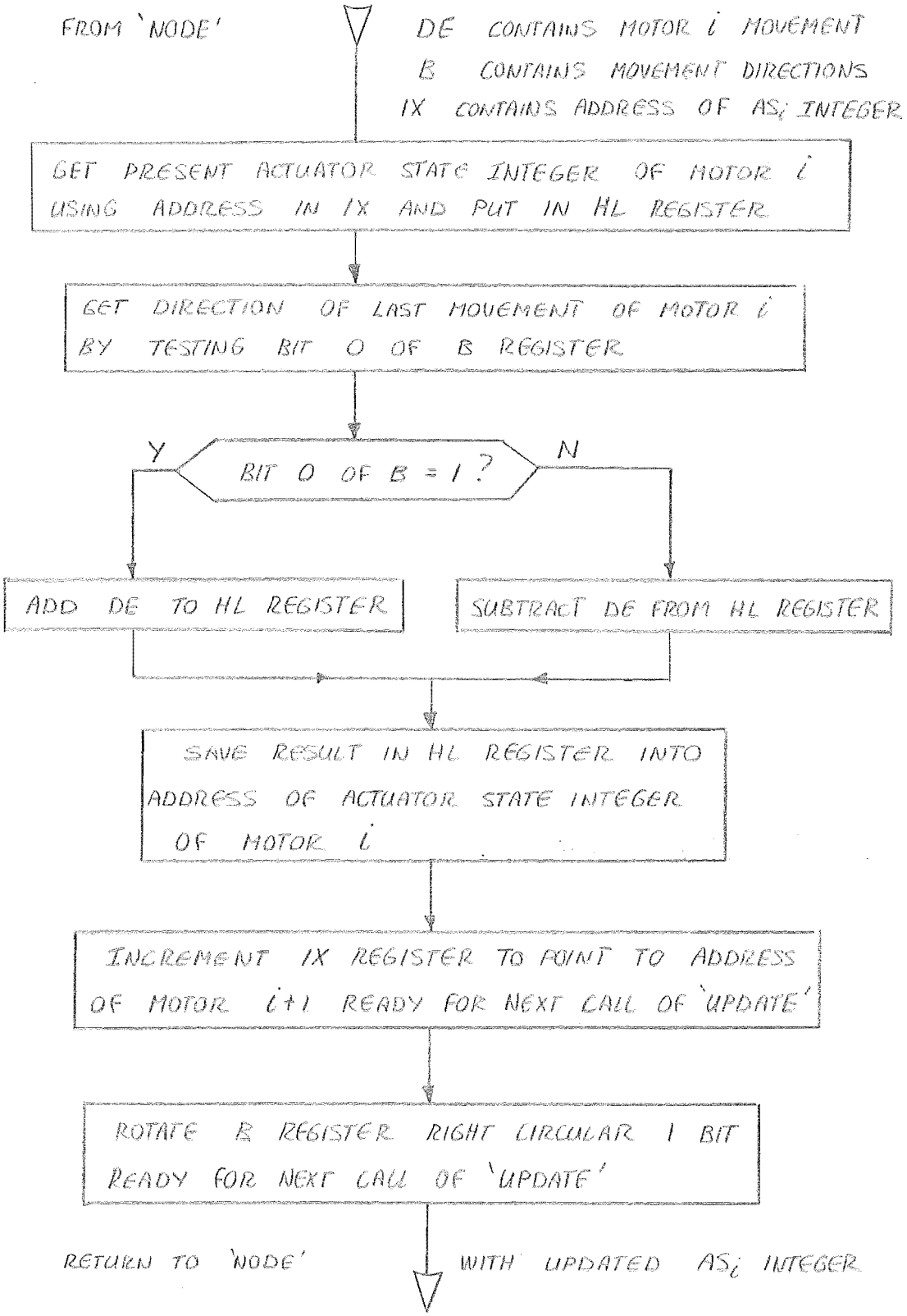


Figure 5.20 The UPDATE program.

CHAPTER 6

ACCELERATION CONTROL

6.1 Motor Acceleration Control Strategy

KODE generates N System Vectors to span a Path Vector. These KODE generated System Vectors have almost the same movement values and the same speed divisor. Thus, the motor speeds (in steps/sec) are almost constant along a KODE generated Path Vector.

However, we would expect the motor speeds to change as the movement passes a Path Vector node. This is because of three effects that can superimpose to give a large change of motor speeds at such a node. These are:

(a) The non-linearity of the six-space to leg-space relation means that motor speeds must change, even if the actual platform motion is of constant velocity. These speed changes only occur at Path-Vector nodes. So, the further apart these are (due to a low commanded resolution for example), then the greater will be the likely motor speed changes across them.

Other effects can occur at Control vector (MIU) nodes. These effects are due to changes in either or both speed and direction of movement in six space and are:

(b) A change in feed speed along a movement will increase the motor speed changes across an MIU node by the same ratio as the feed speed change across the MIU node.

(c) A change in direction of platform movement will cause a large re-distribution of speeds and directions among the six motors, even if the speed divisor remains unchanged.

All these effects can lead to very large motor accelerations across Path Vector nodes that would limit the maximum motor speeds to low values (< 100 steps/sec) if some form of velocity control was not implemented.

A pre-calculated acceleration control is used. It works using the fact that a Path Vector is comprised of many (about 10) almost equal System Vectors.

The speed divisors in these System Vectors can be altered after they are produced by KODE so that a control over motor accelerations along a Path Vector can be in say, 9 motor speed changes at System Vector nodes. These must be chosen so that the overall speed changes between Path Vectors are obtained.

Three important parameters in this acceleration control strategy are now introduced.

(a) VT: VT is called the target speed and applies to the fastest motor. It is the speed that KODE desires the fastest motor to move at and is the value used to calculate the speed divisor in the unmodified MCDB. A file composed of unmodified MCDB having VT less that about 100 steps/sec will in fact perform satisfactorily.

(b) VE: the end of every Path Vector there will be an inevitable change in relative speed divisors due to conditions (a) and (c) mentioned at the start

of this chapter. Even if the speed divisor remains unchanged as movement passes this node there will be certain accelerations, decelerations and perhaps direction changes among the motors. Accordingly there is a maximum speed (of the fastest motor) at which one or more motors will be at their acceleration limit. (Keeping the speed of the fastest motor the same).

This speed is called VE for the maximum allowable speed of the fastest motor in the last System Vector of a Path Vector.

(c) VP: VP refers to the present speed of the fastest motor in any System Vector.

6.1.1 Acceleration Limitations

The limitations on motor accelerations used herin are empirical in nature but based on approximations to theoretical acceleration curves and also on experimental verification. Theoretical methods used to obtain the allowable acceleration and deceleration profiles can be found in reference [5], section 6.3. They are:

(a) For Acceleration: A simple approximation to the correct acceleration profile is to use an exponential type curve. In our case an $x^{\frac{1}{2}}$ curve is used as below,

$$N \geq \frac{1}{K_a^2} (VT^2 - VP^2) \qquad (6.1)$$

Which gives the number of steps on (or System Vectors in this case) over which an acceleration from VP to VT can take place.

A value of $K_a = 158.11$ enables an acceleration from VP = 0 to VT = 500 steps/sec in 10 System Vectors.

(b) For Deceleration:

$$N \geq \frac{VP - VT}{K_d} \qquad (6.2)$$

a value for K_d of 100 enables a deceleration from VP = 500 to VT = 0 in 5 System Vectors.

(c) For a Direction Change between System Vectors k and k + 1:

$$|V_k| + |V_{k+1}| \leq 150 \qquad (6.3)$$

There is one other consideration that greatly simplifies the acceleration control programs. This is the fact that only at Path Vector nodes do accelerations of other than the fastest motor need to be considered. All other nodes are generated by linear interpolation in leg space and thus have only small accelerations produced because of the discrete nature of the movement. Hence, it is only at Path Nodes that the accelerations of all motors need to be monitored to produce a value of VE.

From the above considerations it was realised that a speed control program could process a raw file of MCDB by processing each Path Vector sequentially. The input to

process a Path Vector would be:

(a) The location of the Path Vector in the file.

(b) The length (number of System Vectors) of the Path Vector (N).

(c) The speed at which the Path Vector is entered (VP).

(d) The desired speed in the Path Vector (VT).

(e) The maximum allowable exiting speed from the Path Vector (VE).

(f) Relations giving the allowable accelerations and decelerations within the Path Vector.

Given this data there are 6 possible acceleration events that can take place along a Path Vector. These are shown in Figure 6.1.

Of these six events (A, B, C, D, E, F) only event F has no conditions imposed on its execution. The limitations to the other five events are now described.

Event A

The maximum allowable target speed VT_{max} is obtained when deceleration begins immediately after the maximum speed is reached. We have, from equation 6.1,

$$BA = \frac{1}{K_a^2} (VT^2 - VP^2) \qquad (6.4)$$

And from equation 6.2,

$$\text{and } BB = \frac{VT - VE}{K_d} \qquad (6.5)$$

And we require that,

$$BA + BB = N - 1 \qquad (6.6)$$

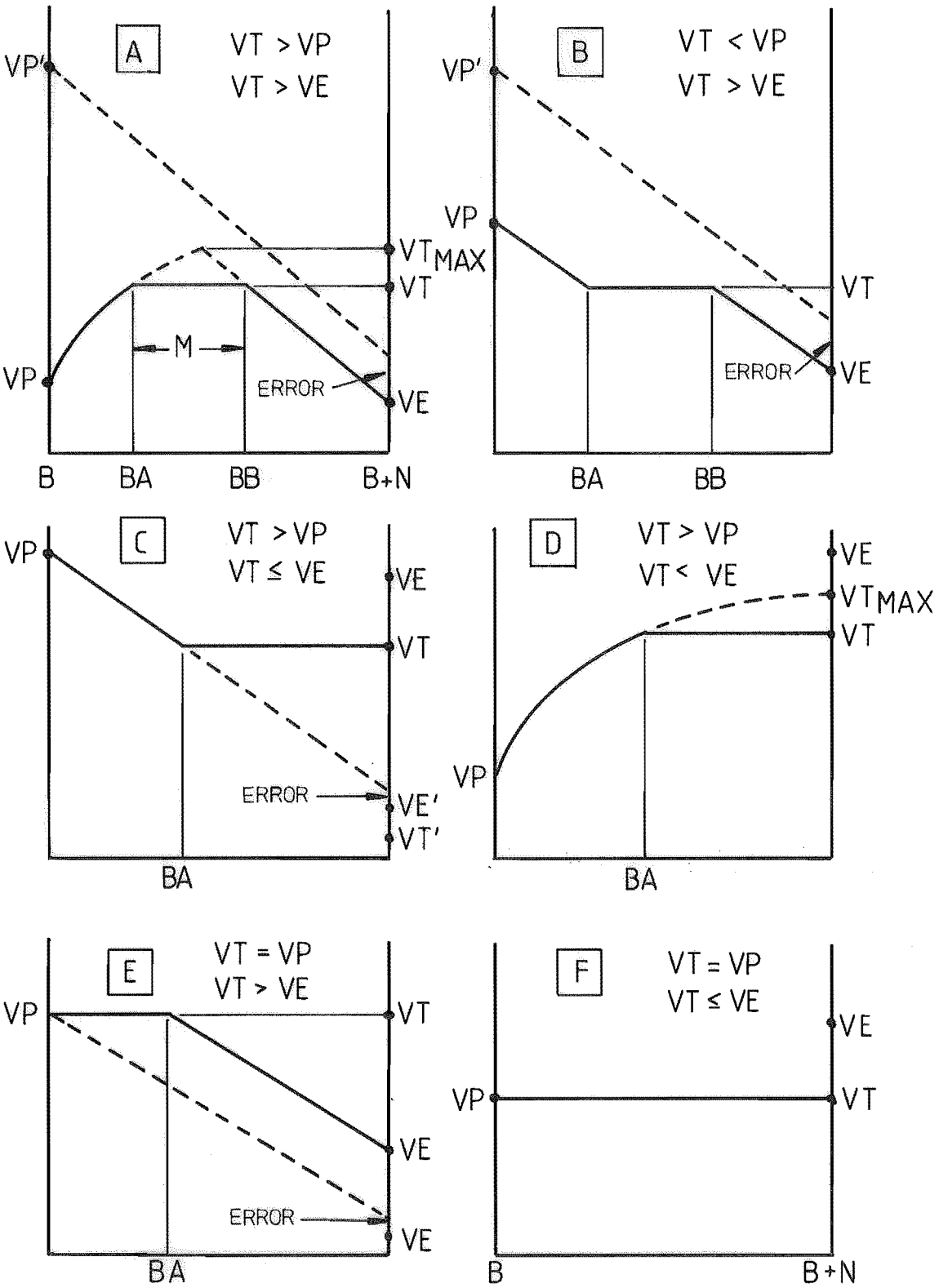


Figure 6.1 Six possible motor acceleration events.

Allowing 1 block spare for "safety". These equations yield a quadratic equation in VT where $VT = VT_{max}$. The solution for VT_{max} is:

$$VT_{max} = \frac{-\frac{1}{K_d} + \sqrt{\left[\frac{1}{K_d}\right]^2 + \frac{4}{K_a^2}\left[N-1 + \frac{VP^2}{K_a^2} + \frac{VE}{K_d}\right]}}{2/K_a^2} \qquad (6.7)$$

where the positive value of VT_{max} is taken.
If the target speed VT is greater than VT_{max}, then VT must be "held" to VT_{max}.

An error can occur if the value of VT_{max} is less than VP. This indicates that VE cannot be reached even if deceleration begins immediately.

Event B

An error can occur here if VE cannot be reached even if deceleration occurs over the whole Path Vector. The error arises if the number of System Vectors is less than NMIN where:

$$NMIN = \frac{VP - VE}{K_d} + 1 \qquad (6.8)$$

Otherwise, VT can take any value between VP and VE.

Event C

The same error condition as event B applies.

Event D

Here, VT < VE so the finishing requirement is satisfied automatically. However, there is a limit to the maximum speed VT_{max} which may be lower than the desired speed VT, where:

$$VT_{max} = \sqrt{K_a^2(N-1) + VP^2} \qquad (6.9)$$

If $VT_{max} < VT$, then VT must be "held" to VT_{max}.

Event E

The same error condition as events B and C applies.

6.2 Motor Acceleration Control Software

The software used to control motor accelerations is comprised of the subroutines described below.
Their operational relationship is shown in Figure 6.2.

(a) SPDDAT The SPDDAT flow chart is shown in Figure 6.3. The purpose of SPDDAT is to generate an array, called SPD that contains data essential to the operation of the aforementioned acceleration control strategy.

SPDDAT is called from KODE at the beginning of the calculation of each new Path Vector. It calculates and records:

(i) The number of System Vectors in the last Path Vector (N),

(ii) The maximum allowable speed (VE) that the last Path Node can be traversed at.

If this data was not saved as it is produced, a subsequent acceleration control program would require a very complex method of extracting this data from the MCDBs being processed.

(b) SPEED: SPEED is the program that controls the acceleration control processing of a raw file of MCDBs (see Figure 6.5).

Input data comes from the SPD array (containing N and VE for every Path Vector), and from the

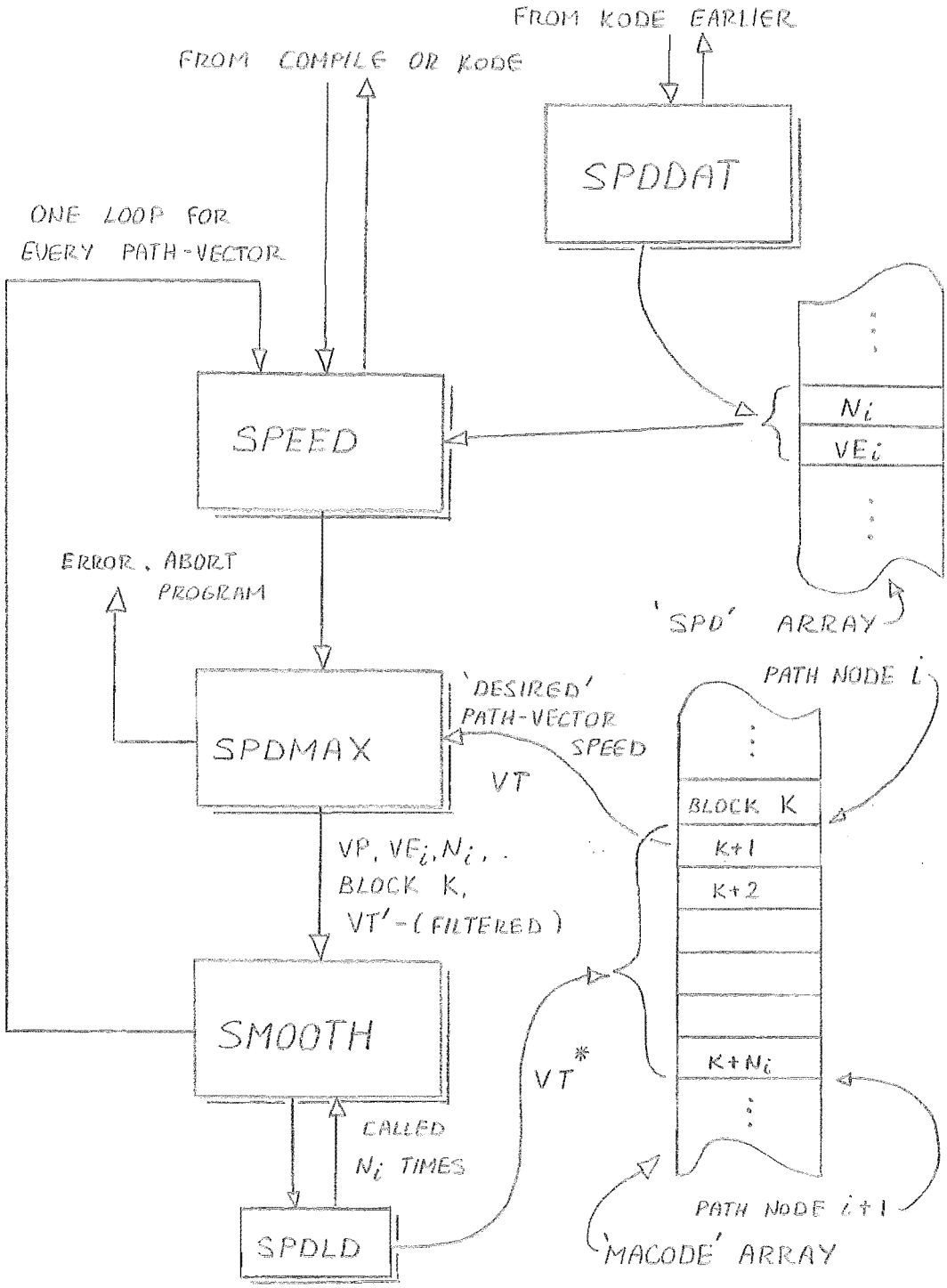


Figure 6.2 Speed processing operations.

MACODE array which contains the MCDBs which each have a target speed (VT).

The output data is the MACODE array, unaltered except for modified values of VT in many of the MCDBs.

Subroutines within SPEED update and keep track of the other relavant variables which are:

(i) The present location within the MACODE array, (BLOCK).

(ii) The present speed of the fastest motor (VP).

SPEED merely controls the progression of the Path Vector based acceleration control strategy through the Path Vectors in the MACODE array. It introduces special functions that are necessary at the starting and finishing Path Vectors.

For every Path Vector, SPEED calls the following trio of subroutines which do the actual acceleration control on the particular Path Vector.

(c) SPDMAX (See Figure 6.6) SPDMAX is called from SPEED. SPDMAX determines from the input data which of the events A, B, C, D, E or F apply. It then checks that there are no errors and/or modifies the target speed VT so that acceleration limitations are not exceeded. Any error will cause an abort of the whole COMPILE program. Control then passes to SMOOTH which handles the actual modifications to the System Vector speeds.

SPDMAX can be thought of as a filter that only allows SMOOTH to be called with correct data.

(d) SMOOTH (See Figure 6.7), given correct data, can modify the speed divisors in the N System Vectors of a Path Vector to achieve the desired acceleration events as shown in Figure 6.1.

The actual modification of speed divisors is done by the simple subroutine SPDLD (see Figure 6.8) which loads the value commanded by SMOOTH into a block and serves to increment the BLOCK value each time it is called.

The following flowchart descriptions of the SPDDAT, SPEED, SPDMAX, SMOOTH and SPDLD programs are largely self explanatory when analysed in conjunction with Figure 6.2.

Some operations occuring in these subroutines need extra explanation which is given below. Any further operational details can be extracted from the program listings given in Appendix 3.

SPDDAT Flow Diagram Explanations

(Referring to Figure 6.3)

Function A: On the first call of SPDDAT which corresponds to the starting node, there is no preceeding Path Vector so A detects this and routes the program through B to E.

Function B: Puts the Path Vector indice k to 1, and puts the BLOCK offset, OFFSET to 0.

Function C: Detects whether a new OFFSET value has appeared. This indicates that the previous data in SPD has been used. Thus new data can be placed at the start of SPD

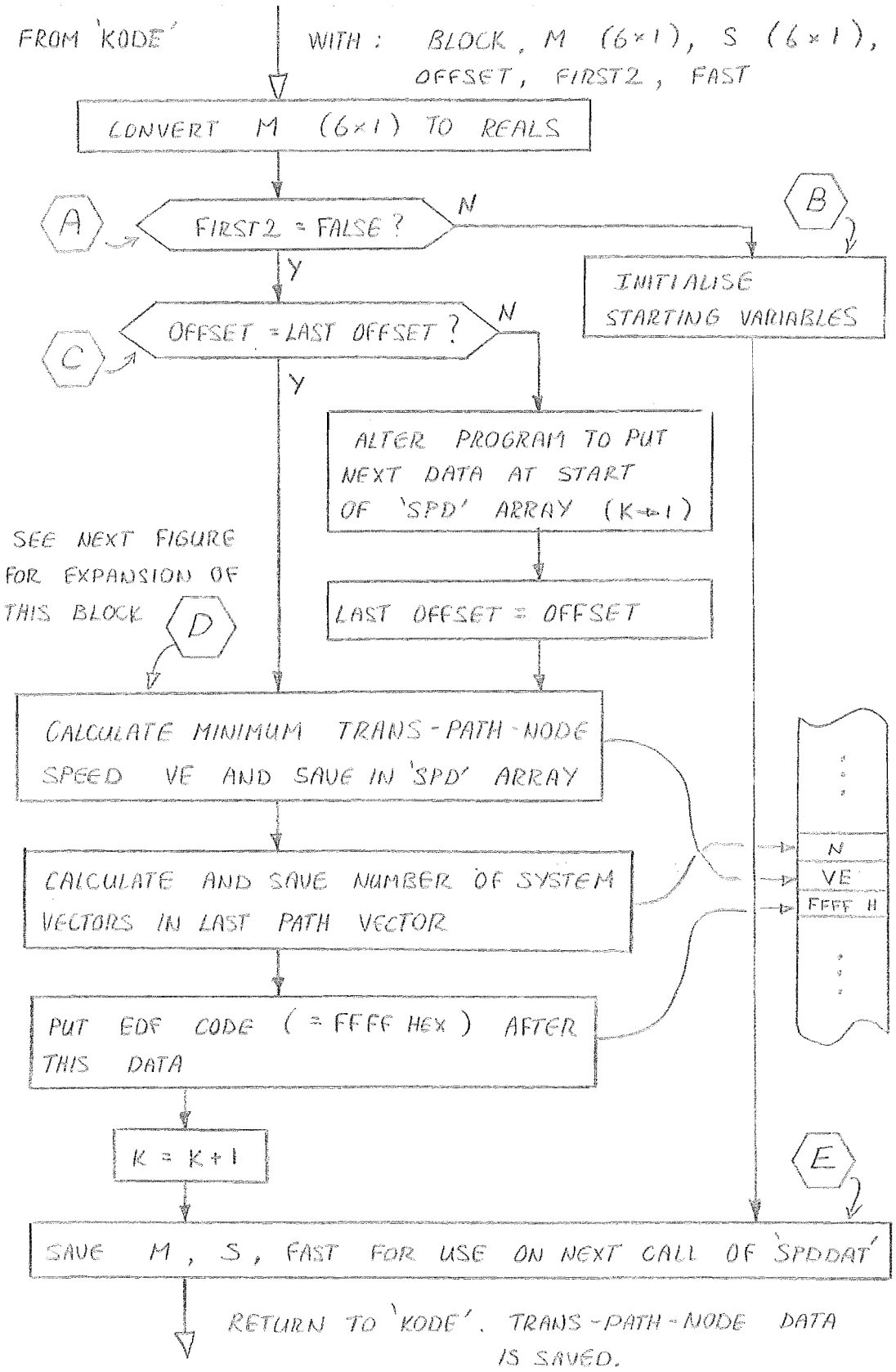


Figure 6.3 The SPDDAT program.

again, overwriting "used" data. Thus K is set to 1 and OFFSET is put into LSTOST ready to sense the next change of OFFSET.

Function D: calculates the maximum allowable trans-node speed of the fastest motor. This calculation assumes the speed of the fastest motor is the same before and after the node. Function D is shown in Figure 6.4

Accelerations due to speed changes are calculated using:

$$A_i = \frac{M(i)}{M(FAST)} - \frac{LASTM(i)}{LASTM(LSTFST)} \qquad (6.10)$$

where: A_i, when multiplied by VP gives the change in speed of motor i across the Path Node,

M(i) = the movement (steps) of motor i in the present System Vector (Post-Path-Node),

LASTM(i) = the movement (steps) of motor i in the last System Vector (Pre-Path-Node),

FAST = the fastest Post-Path-Node motor,

LSTFST = the fastest Pre-Path-Node motor.

Calculation 6.10 is performed by Function F. Thus, six values of A_i are generated. The greatest negative value of A (if any) and the greatest positive value of A (if any) enable us to calculate two acceleration limits, called VEA and VED respectively.

VEA is calculated by using Equation 6.1. This equation enables us to derive the maximum speed at which a positive acceleration can take place.

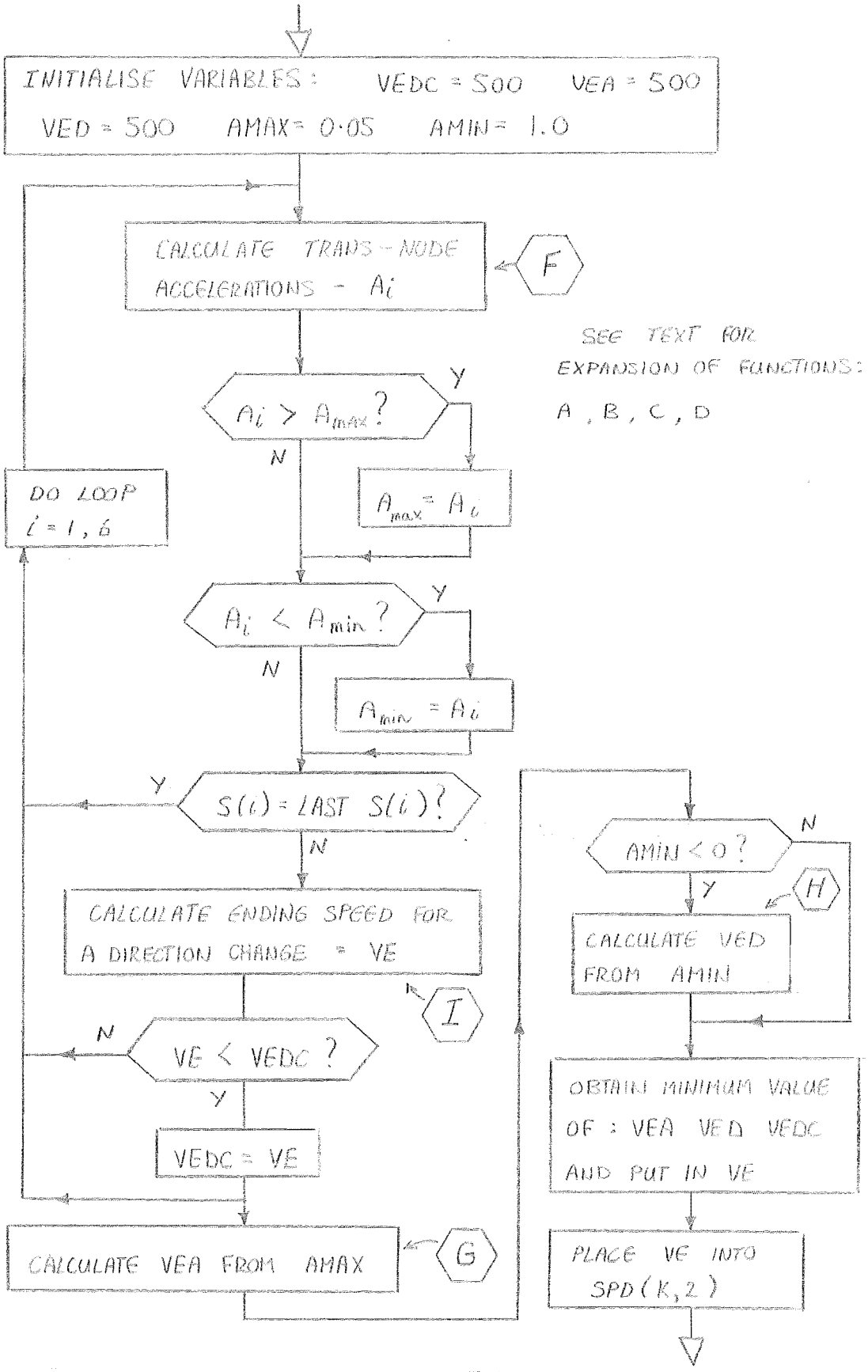


Figure 6.4 Function D of SPDDAT (from Figure 6.3)

From 6.1 we have:

$$\frac{VT^2 - VP^2}{K_a^2} = 1 \qquad (6.11)$$

Also, from equation 6.10 we have:

$$VT = (1 + A)VP \qquad (6.12)$$

Substituting this into the above equation eliminates VT and enables us to calculate the maximum value of VP. We obtain:

$$(1 + A)^2 VP^2 - VP^2 = K_a^2$$

Giving

$$VP = \frac{K_a}{\sqrt{A^2 + 2A}} \qquad (6.13)$$

Thus, putting the greatest positive value of A (called AMAX) into equation 6.13 gives us the limiting positive acceleration speed denoted by VEA. This is done by function G in Figure 6.4.

VED is calculated using equation 6.2 with N = 1, and equation 6.12. These give:

$$VP = \frac{K_d}{A} \qquad (6.14)$$

Thus, putting the greatest negative value of A (called AMIN) into equation 6.14 gives us the limiting negative acceleration speed denoted by VED. This is done by function H.

The remaining possible acceleration is that due to a direction change, and the corresponding maximum allowable speed of the fastest motor is denoted by VEDC. Equation 6.3 gives the limitation. That is:

$$V_k + V_{k+1} = 150 \quad \text{with } N = 1$$

Now

$$V_{k+1} = \frac{M(i)}{M(FAST)} \cdot VP$$

And

$$V_k = \frac{LASTM(i)}{LASTM(LSTFST)} \cdot VP$$

Giving

$$VP = \frac{150}{\frac{M(i)}{M(FAST)} + \frac{LASTM(i)}{LASTM(LSTFST)}} \qquad (6.15)$$

This function is done by function I only if there is a direction change.

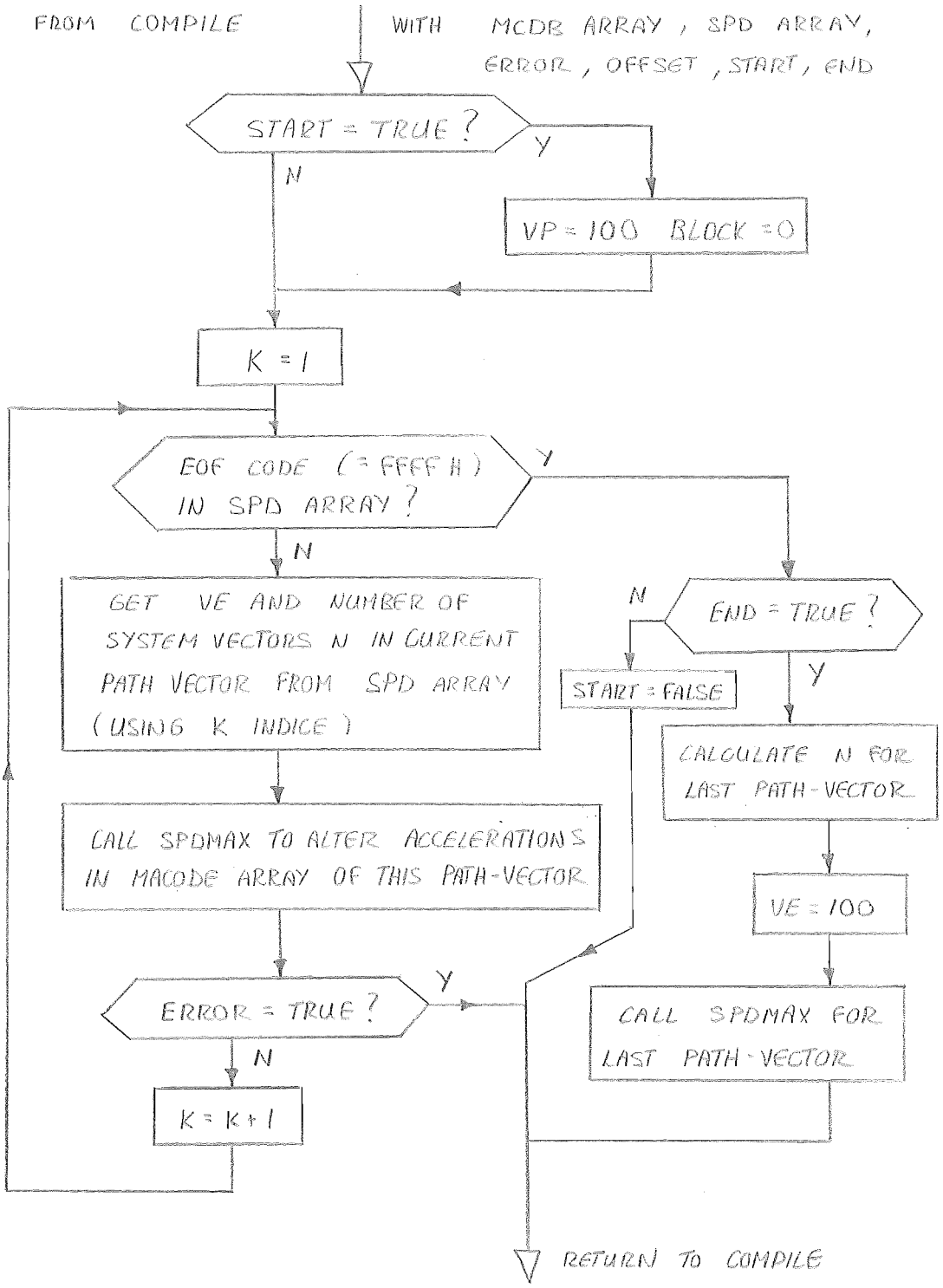


Figure 6.5 The SPEED program.

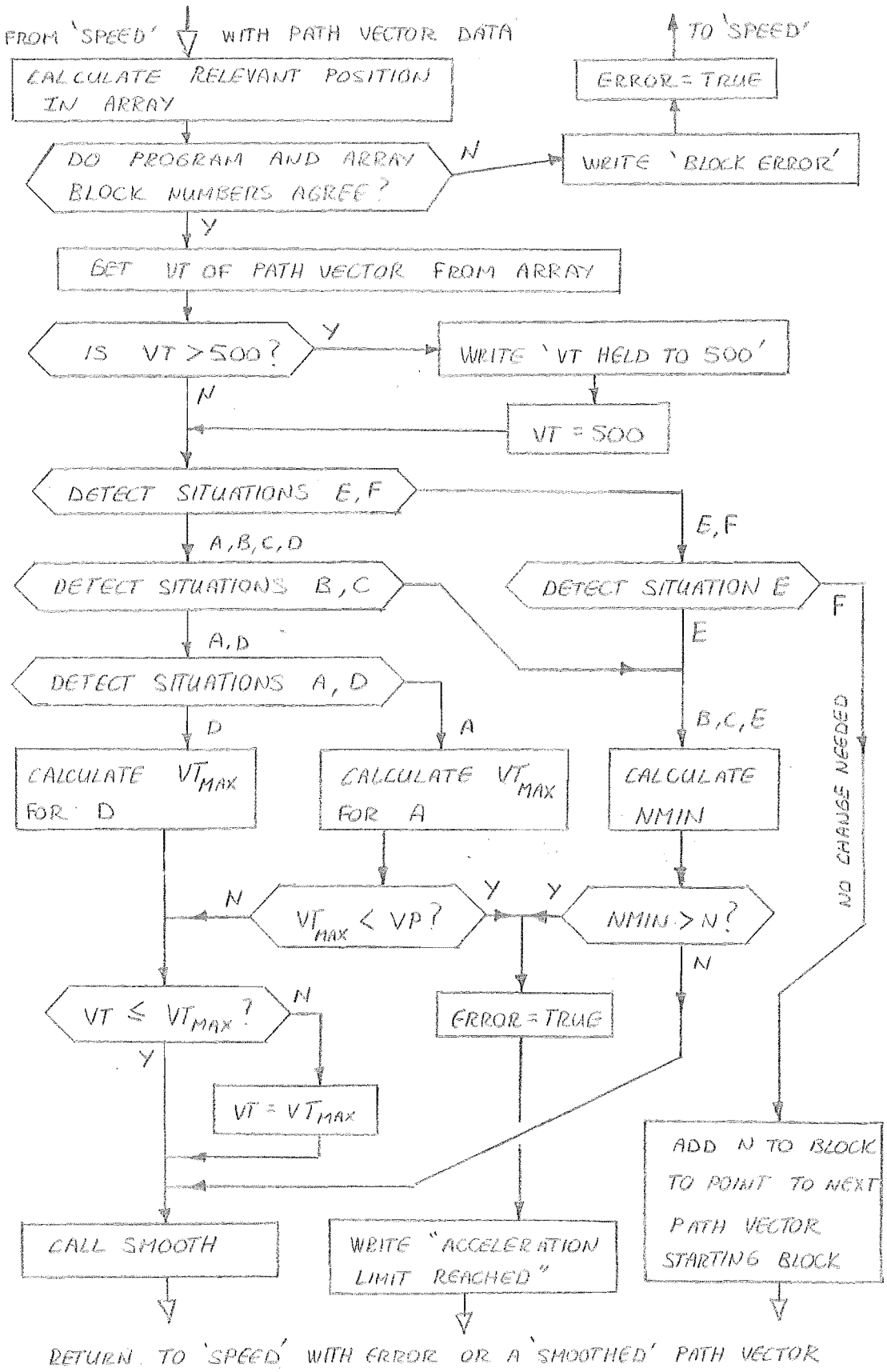


Figure 6.6 The SPDMAX program.

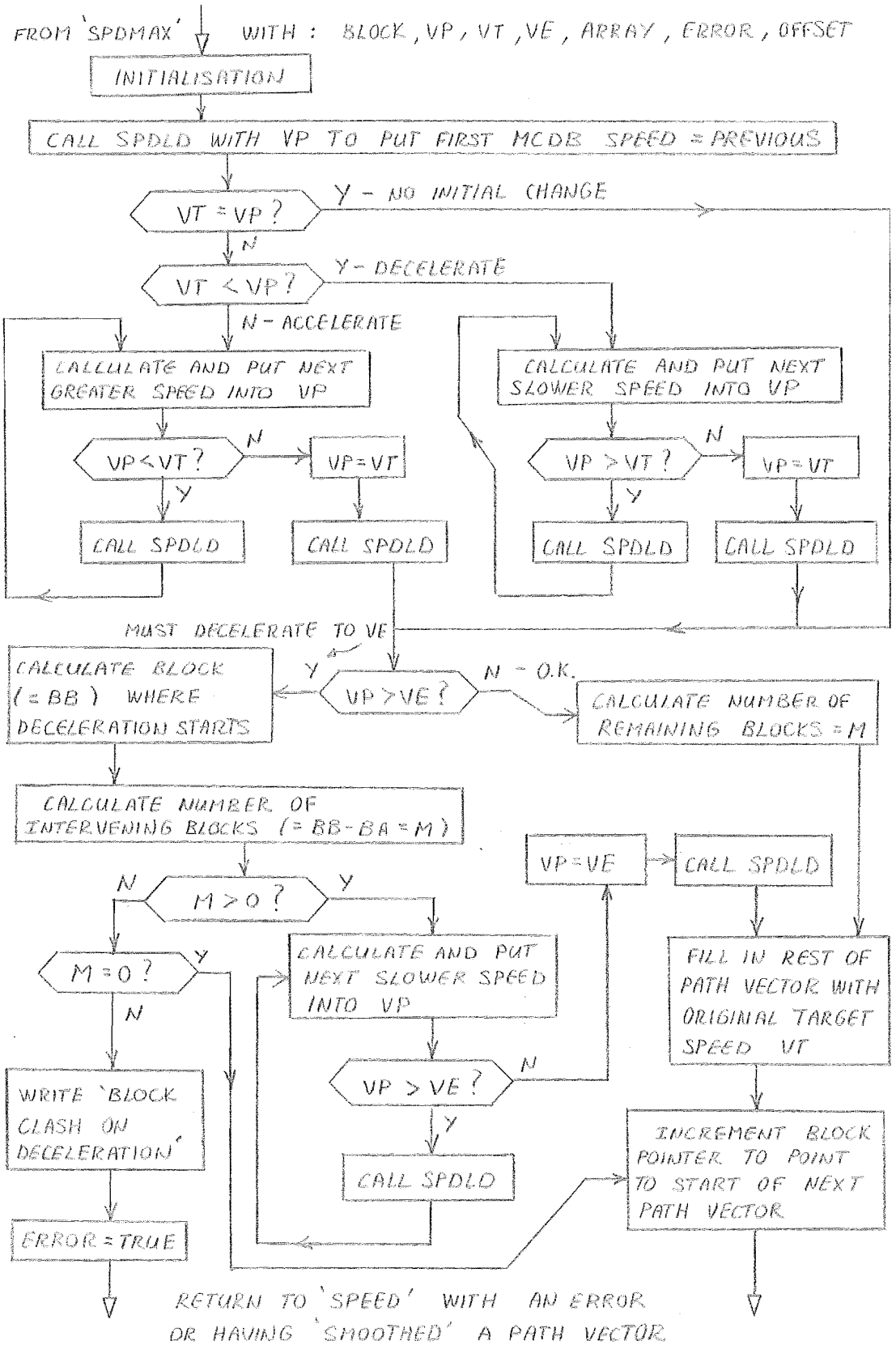


Figure 6.7 The SMOOTH program.

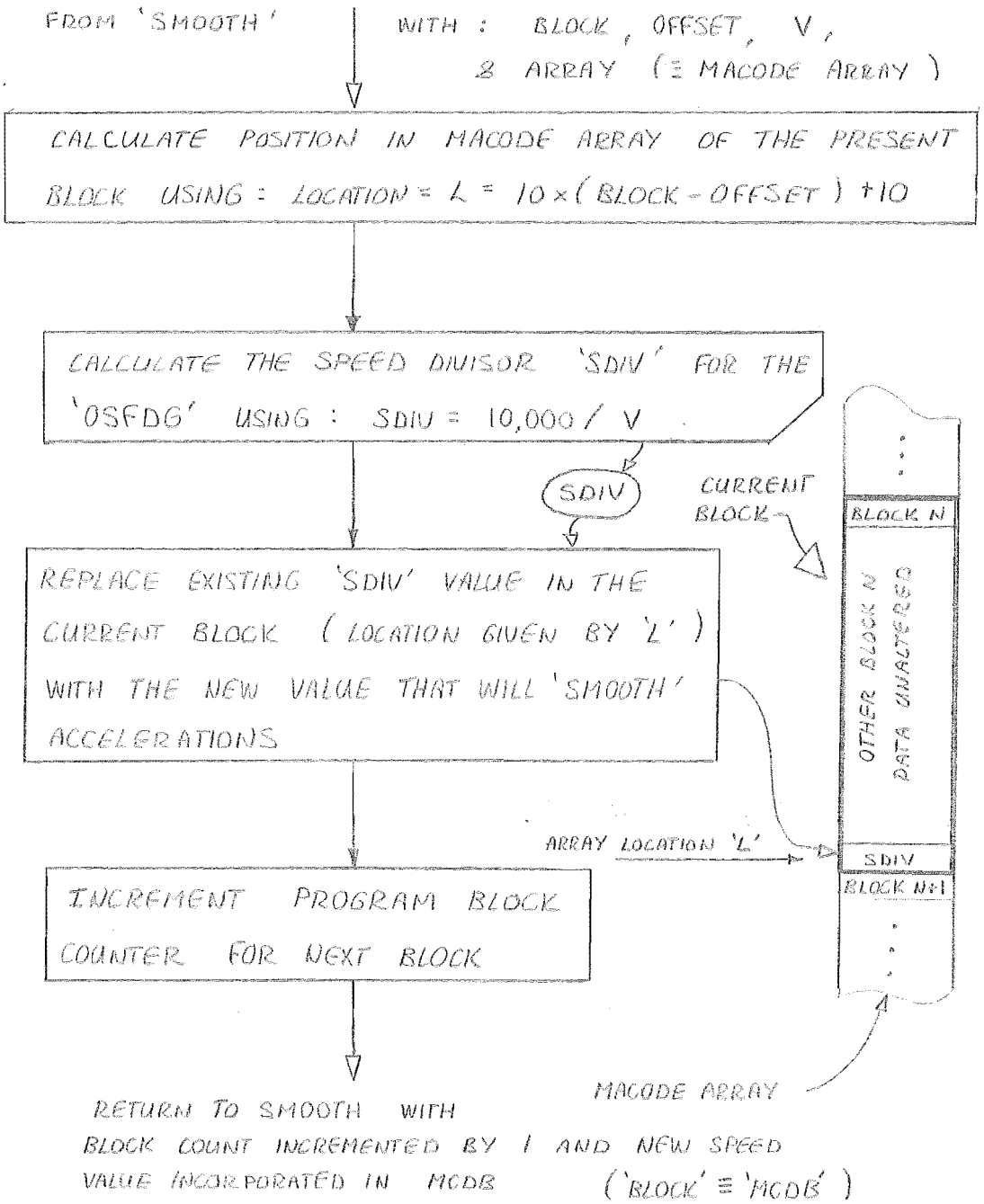


Figure 6.8 The SPDLD program.

CHAPTER 7

OPERATION

7.1 General Operation

The University of Canterbury Stewart Platform can be controlled entirely by an operator sitting at the computer console shown in Figure 1.2.

The sequence of activities needed to make the Stewart Platform perform a desired movement are described below:

(1) The operator should compose the desired movement according to the input format described in section 5.1.1, and record this, plus a tool path diagram in an "operations notebook".

The following points should be noted:

(a) Coordinates ("C" lines) describe the position of the tip of the platform zero vector ("P" lines), and the orientation of the platform axis system, with respect to the reference axis system X, Y, Z shown in Figure 3.4.

(b) The minimum height of the platform is at about a height of Z = 270mm above the Hooke joint axes.

(c) The orientation of the platform zero vector is described by the orientation of the platform axis system x, y, z, in euler angles.

(d) Feed speeds ("V" lines) apply even if only changes in orientation occur. Then, the feed speed is taken as the speed of movement of one of the platform corners.

(2) The Computer System can now be turned on. To do this, the power leads to the : computer main board, VDU, expansion unit and expansion unit auxiliary supply unit must be turned on simultaneously.

Note that the data on a floppy disc may be damaged if a voltage fluctuation occurs in the computer system while the door of that floppy disc drive is closed. Such voltage fluctuations can occur whenever a mains circuit is switched, so keep the disc doors open whenever this occurs.

(3) Insert a "NEWDOS" operating disc into the left hand (drive 0) disc slot, and the "STEWART PLATFORM" operating disc into the right hand (drive 1) disc slot and close both drive doors.

(4) Press the computer reset button to "boot" the computer into the NEWDOS operating system. A NEWDOS logo should appear on the screen.

(5) Now the SOURCE file can be created, using the EDIT program to create and save a file of the information composed in step 1.

Filenames are limited to six literal characters.

(6) Now, the SOURCE file must be converted into a RUN file using the COMPILE program.

COMPILE is called and will display a menu. Select the "COMPILE A FILE" option.

COMPILE will then ask for the SOURCE filename and the desired RUN filename. It is suggested that RUN filenames be the SOURCE filenames with an "X" letter tag to differential between SOURCE and RUN files for a particular movement. Files can be overwritten.

COMPILE will then "compile" the SOURCE file into a RUN file. COMPILE's progress is indicated by block numbers appearing on the screen.

Fatal and non-fatal error may occur. These are described below.

Fatal COMPILE Error Messages

All fatal errors cause a return to the COMPILE menu. They give the operator some information on the nature of the error.

The data on the RUN file is saved in "chunks" of about 450 MCDBs. If a fatal error causes the COMPILE program to "crash", the RUN file may contain usable MCDBs if the error occurred after the first data "chunk" was saved. This file can be run by the CCS and may show why the error occurred.

The fatal error messages are:

(a) "LEG i MAXIMUM LENGTH PASSED" or "LEG i MINIMUM LENGTH PASSED". These messages indicate that leg i has reached its travel limit. This information can be used to adjust the SOURCE file to eliminate the error.

(b) "BLOCK # ERROR" Indicates that a block number mismatch has occurred between the MCDB data and the SPEED program. This message was used for checking the operation of the program during development. It should not occur during SOURCE file processing.

(c) "CANNOT ACCELERATE IN # OF BLOCKS" Indicates that SPEED has detected a Path Vector which has insufficient MCDBs (= System Vectors) to allow the Path Vector ending speed VE to be reached from a higher speed.

The solution is to reduce the commanded feed speed over this area of movement. This reduces the deceleration required to reach VE.

(d) "FILE ERROR" indicates that the SOURCE file format is not correct. Incorrect SOURCE file format can be caused by non-compliance with the format specification of Figure 5.5 or because the SOURCE file is too large, overflowing the input buffer array.

Warning COMPILE Error Messages

These messages are generated during the SPEED processing of the raw MCDBs. They indicate that actuator speeds are being held to certain limits. These messages are:

(a) "VT HELD TO 500" Indicates that a commanded actuator speed of VT is being held to the maximum allowable speed of 500 steps/sec.

This indicates that the movement speed is too high.

(b) "VT HELD TO VT MAX IN BLOCK N" Indicates that a commanded actuator speed of VT has been held to a calculated speed of VT MAX so that the Path Vector ending speed VE can be reached. This indicates that the acceleration/deceleration limits have been reached.

A RUN file having these warning messages will run satisfactorily but some commanded speeds will not be reached.

Once a satisfactory RUN file is produced, it may be used to provide the desired motion of the platform.

(7) Switch on mains power to the six Actuator Control Units (ACUs), ensuring disc drive doors are open. The ACUs dissipate a large amount of heat in the motors and series resistors and thus should be turned off during long idle periods.

(8) Call up the Computer Control System by calling the CCS program from NEWDOS. This program is menu driven.

(9) As the CCS program has just been loaded, the platform must be zeroed to establish a known position.

Select the "ZERO THE TABLE" program. This program asks for a zeroing speed and then proceeds to move the platform to its zero stops.

The zeroing speed should be less than 300 steps/sec. The operator can stop any motor which has reached its stop by pressing keys 1 to 6 for that motor.

Pressing "BREAK" stops all motors and returns to the CCS menu.

(10) Once the table has been zeroed, select the "RUN A MACHINE CODE FILE" program. This program asks the operator for the following information:

(a) "RUN FILENAME?". Enter the desired RUN filename. The CCS will load the first 12 bytes of the file into memory.

(b) "STARTING FROM ZERO? Y/N" Always answer "Y" unless the platform has just undergone a controlled movement and the CCS program remains loaded.

(c) "STOP AT NODES? Y/N" Answering "Y" causes the controlled movement to stop at each System Node. This is useful for MCU control development as it enables the MCU operation to be easily investigated. Commanded speeds should be low. Otherwise answer "N".

(d) "SPEED TO START POINT? STEPS/SEC". The operator should input an actuator speed less than 200 steps/sec.

(e) Once the movement to the start point has been calculated this question is asked to give the option of the exit back to the MENU:

"READY. G-GO, X-EXIT"

Entering "G" will cause motion to the start point to commence.

The operator can stop, restart and exit from the movement using the "SPACE", "ENTER" and "BREAK" keys as described on the VDU.

(f) The platform will stop when the starting position has been reached whereupon this message will appear:

"STARTING POSITION ACHIEVED,
LOADING REST OF FILE".

(g) When the rest of the file has been loaded the operator is given the choice of starting the movement or returning to the MENU by the question:

"READY. G-GO, X-EXIT"

Entering "G" starts the commanded movement of the platform.

(h) The platform executes the movement and when completed the platform is stopped and the program returns to the MENU.

Subsequent movements can be executed by the same sequence of (a) to (h) except that a "N" answer is given to question (b).

Once the desired movements are completed the operator can return to NEWDOS by selecting the "EXIT TO NEWDOS" menu option. The present position of the platform will be lost, and the platform must be re-zeroed next time the CCS is loaded.

The shutdown procedure is as follows:

(11) Open the disc drive doors and turn off the six ACUs.

(12) Turn off the computer equipment described in step 2 simultaneously.

(13) Place the floppy discs in a safe place.

(14) Record notes on the movement(s) in the "Operator's Notebook".

7.2 Milling Operation

The Stewart Platform has been equipped to enable experimental milling operations on polyurethane foam. The modifications to the basic platform to achieve this were:

(a) A cutting head (DeSoutter pneumatic drill) was attached to a solid member above the platform. It was roughly positioned to be in-line with the Z axis of the reference coordinate system. The position and orientation of the cutting head can be controlled by varying the clamping attachments and/or the drill's integral down-feed mechanism.

(b) A special tool was constructed that could cut along any tool path and take wide cuts to minimise tool path travel. Tool path travel is limited by memory storage limits.

The tool is effectively a 35mm slot drill for light cutting duties with a removable cutting blade.

(c) Two solenoid valves were connected to the cutting head allowing it to be turned on or off using a

control pad at the operator's console.

NOTE: These solenoid valves can produce line voltage fluctuations when switched.

7.2.1 Milling Experiments

Several milling operation experiments were done to check the operation of the system and to assess the usefulness of having control over the six degrees of freedom of the platform.

The operations were performed on a rigid block of foam attached to the platform by a removable workpiece holder. The operations were:

(a) A movement file was created that would mill a flat surface on the foam block of dimensions 80mm by 80mm square, at a feed speed of 5 mm/sec.

The operation was performed successfully and the surface was found to be flat (using a straightedge) in two orthogonal directions (see Figure 7.1).

Measurement accuracy was limited to about 0.1mm due to the non-homogeneity of the foam.

This experiment indicated that the whole geometric/control system was working satisfactorily.

(b) The same movement as (a) was executed on a plane at 10 degrees to the horizontal. This was done by altering the second euler angle.

FIGURE 7.1

Milling a Flat Surface

of movement (a) from 0 to 10 degrees. Once again the surface was found to be flat.

(c) The cutter was commanded to cut a horizontal, square slot in the foam. Then, the same movement was executed again. The cutter followed the same path, with no extra cutting being detected on either side, or below the cutting tool.

This experiment showed that the repeatability of movement of the platform exceeded that detectable with foam.

(d) To illustrate the capabilities of the platform, a file to cut a cone out of the foam was written. (Appendix 4)

This file commands the platform to move so that the cutter moves in a descending helical path down the cone surface. The cutter axis is always parallel to the cone surface so that the tool creates a smooth conical profile.

When this SOURCE file was compiled, a leg length limit was found to be reached after only one turn of the helix.

The data which had been processed was run to ascertain if the file would produce the correct tool path.

The movement executed 1 turn of the helix and produced a smooth, descending and expanding helical cut in the foam as seen in Figure 7.2.

A cone was starting to form on the inside edge of this cut.

This experiment indicated that the control of all six axes was functioning correctly.

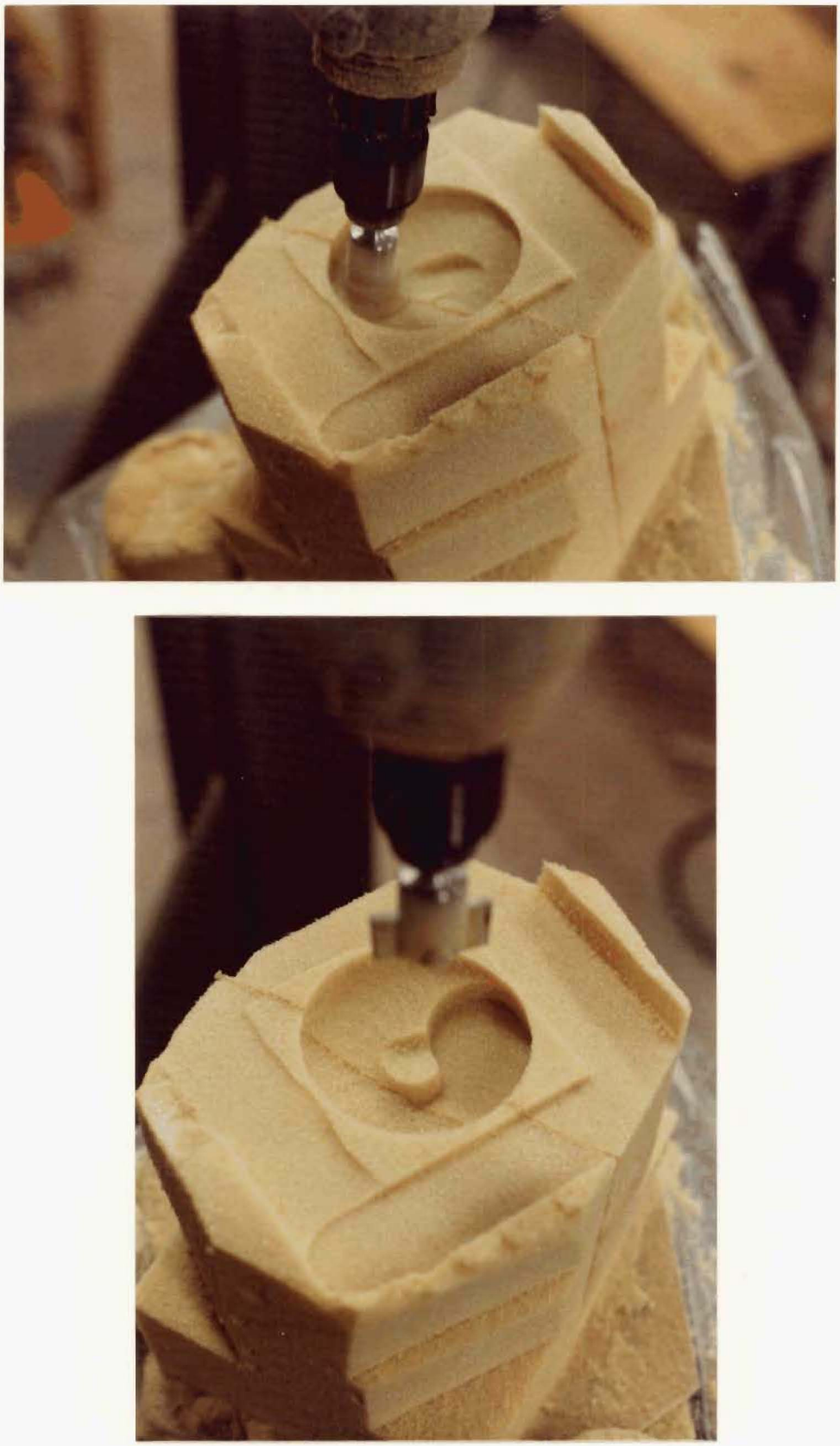


FIGURE 7.2

Milling a Conical Shape

However, the compilation of this and earlier files had shown that the movement envelope of this platform was quite small. Especially so when a combined rotation and translation movement was desired.

CHAPTER 8

A Future System Proposal

The existing Stewart Platform and control system can undertake simple experimental milling and positioning operations. Many improvements are necessary to convert this embryonic system into a fully capable milling machine with six degrees of freedom.

This chapter describes these improvements with the intention that this will guide any further developments in the correct directions.

8.1 Geometric Improvements

A new platform should be constructed using 6 identical discrete legs in the configuration shown in Figure 2.3 (a) rather than the present configuration which is similar to Figure 2.3 (b).

This eliminates a three link joint between the apex of two legs and the platform.

The legs should be joined to the platform and base using simple spherical joints of the ball-and-socket type. This eliminates the complex universal joints used at present at both ends of the legs and simplifies the geometry greatly.

This simplification through reduction of links and joints has three benefits.

(1) The "LENGTH" algorithm that calculates the required leg lengths given the platform position in six space is greatly simplified. This leads to

a simpler, faster and more compact algorithm,

(2) The effect of dimensional uncertainties of the links on accurate positional calculation is reduced,

(3) Free movement, and stiffness reductions through joints are greatly reduced.

In general, there exist many possible platform geometries, many of which may provide an optimum movement envelope for a particular application. Thus, a full movement envelope analysis (probably computer based) should be undertaken to find the optimum configuration for any particular application.

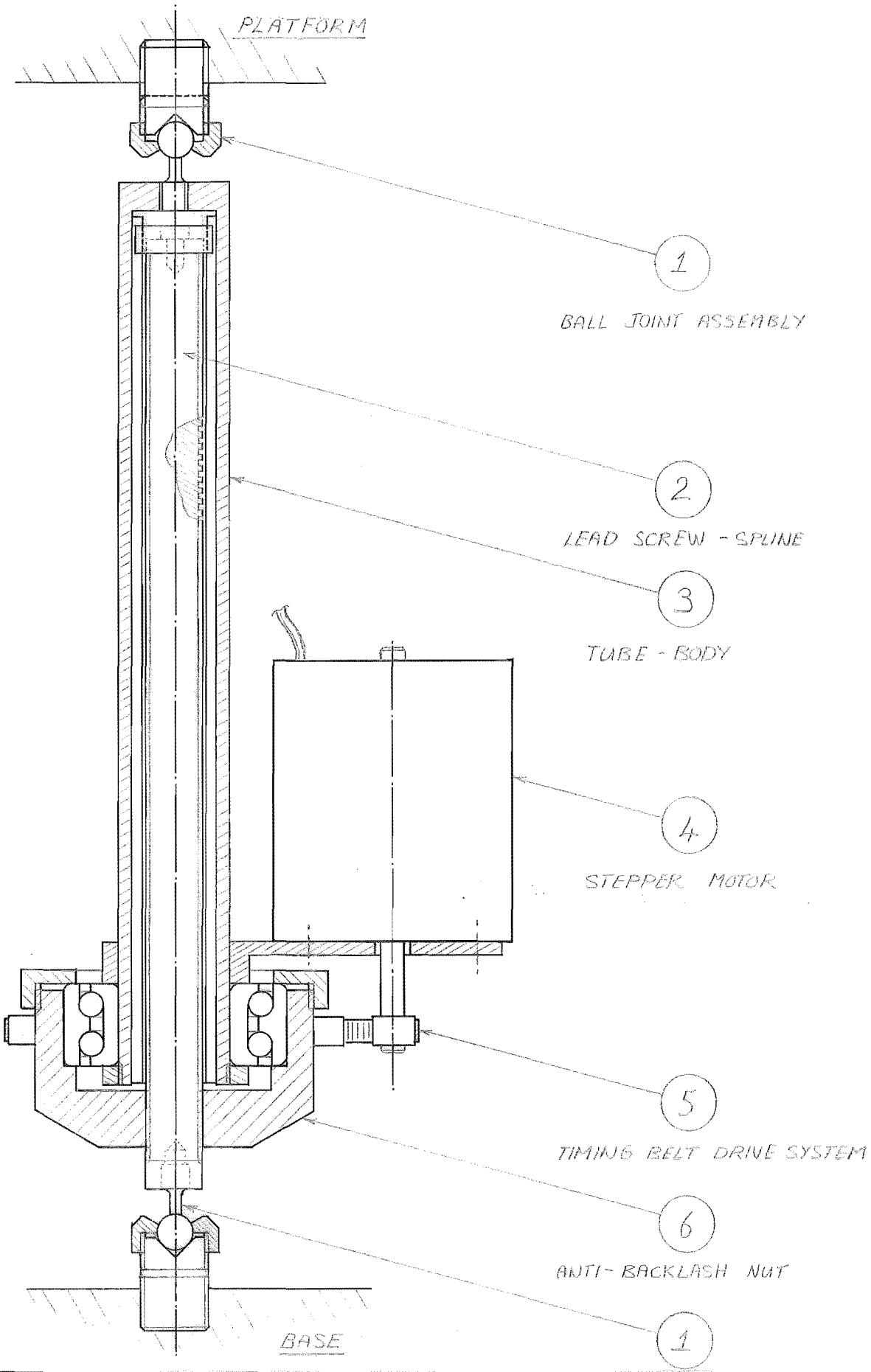
A computer based movement envelope simulation has been done in reference [11]. This work analyses a general platform of the type shown in Figure 2.3 (a) using ball-and-socket leg end joints. This work provides a good starting point to further geometric investigations.

8.2 Mechanical Improvements

The leg lengths should be varied using a high pitch lead screw / actuator system that forms a compact unit between the ball-and-socket joints at each end of the legs. Such a leg is shown in Figure 8.1.

This leg actuator has these special attributes.

(a) The lead screw is a two start power screw having a helix angle of about 8 degrees. This increases the screw efficiency from about 25% at present to about 50% which is just below the reversing limit.



INTEGRAL LEG-ACTUATOR

FOR A STEWART PLATFORM

Figure 8.1	SCALE: 1:2	DRAWN: G.R.		
	DATE: 9-7-85	APPROVED:		

(b) Because of the high lead screw pitch, the anti-backlash nut is driven by the side mounted stepper motor via a timing belt with a speed reducing ratio. This reflects a minimal inertial load onto the stepper motor shaft.

(c) The torque reaction of the lead screw is reacted back to the stepper motor via an internal spline on the motor support tube. This is taken at a reasonably large radius so that loads are low and side play causes the rotational free movement to correspond to less than 1 step of the stepper motor.

(d) The anti-backlash nut is mounted to the motor support tube by a preloaded, double row angular contact ball bearing. This gives a high stiffness rotational joint.

(e) The end mounted ball joints are identical, have a high stiffness in all directions and screw directly into the base or platform. They have a large arc of movement.

(f) The actuators have a reasonably high ratio of maximum length to minimum length.

Attributes (a) and (b) help to keep the torque and inertia loads on the stepper motor to a minimum. This greatly increases the allowable acceleration of the motor.

Attributes (d) and (e) serve to ensure that the leg is adequately stiff both in compression and tension.

Attribute (c) eliminates any motor reaction forces being transmitted to the ball joints.

Attributes (e) and (f) serve to maximise the envelope of movement of the platform.

Overall, a new table incorporating these actuators will have a greater movement envelope, much greater stiffness, faster movement and less cost than the present machine.

8.3 Computer Control System Improvements

The control system performed quite adequately in the tasks it was given, but these following improvements are worthwhile:

(a) A much higher oscillator frequency will allow more steps to be output per block so that a Path Vector may be spanned by a single System Vector. At present many System Vectors are usually required, creating many times more data than is necessary. This improvement requires new frequency dividing hardware.

(b) A control port that allows a command word to be sent to all three STC chips simultaneously would approximately halve the trans-node "dead" time.

(c) A computer having a faster clock speed would reduce this "dead" time further.

(d) Stepper motors should be bi-polar driven at high voltages using "chopper" type current limitation.

(e) A real-time speed control system using two extra interrupt levels could be implemented in software. It would allow extraction of the maximum performance from improvement (d).

(f) Though not strictly necessary, a digital feedback system would increase the system capabilities and positional reliability. Many forms of feedback are possible, including load sensing and shaft position.

8.4 Path Synthesis Improvements

There is almost no limit to the possible expansion of the COMPILE program because it has a "front end" that can incorporate ever higher levels of path synthesis computation.

However, the following improvements should be the first incorporated:

(a) COMPILE (and also CCS) should be run on a much more capable computer with larger memory and faster computation. At present COMPILE takes about 5 minutes to process a file containing about 500 mm of tool path travel. Ten fold speed improvements are easily achievable here, the Z-80 based microcomputer used being extremely slow when compared to contemporary machines.

(b) The only path control type used: "Basic Path Control" should be expanded to include "Circular Path Control".

(c) Cutter geometry compensation should be developed.

(d) Feed speed calculation should be improved for rotational movements.

(e) The resolution calculation procedure should be investigated and developed so that only the

minimum number of intermediate Path Vectors need be calculated for placement along a Control Vector.

8.5 Milling Machine Proposal

Once the above improvements have been developed and proved, the system could be developed into a fully capable 6-axis milling machine with the addition of appropriate cutting head, lubricant systems, etc.

A simple cost estimation of such a system was made and is detailed below.

Cost Estimation for a Stewart Platform Milling Machine

1.	Control System	
(a)	Motors	
	6 Evershed FET2/M55 at $240	$1,440
(b)	Motor Drive Modules	
	6 Evershed C.D. 30 at $982	$5,880
(c)	Power Supply Modules	
	2 Evershed P.M 1200 at $575	$1,500
	(Plus Transformers)	
(d)	M.C.U. Card Estimate	$2,000
(e)	Packaging, Wiring	$500
	Total	**$11,320**

2. Stewart Platform

(a) 6 Leg actuators

Labour: 60 hours each at $20/hr	$1,200 each	
Materials: ½ Labour cost	$ 600 each	
Total -	$1,800 each	
Total for 6 actuators		$10,800
(b) Platform		$1,000
(c) Base		$1,000
(d) Cutting head, tool changer and lubricant system = 1/2 cost of machine		$12,800
(e) Assembly, commissioning period of 1 week		$800
Total		**$26,400**

3. Computer

Current price for high speed processor, large memory PC	$10,000
TOTAL HARDWARE COST	**$47,720**

In comparison, a small conventional 3 axis milling machine having a rotary table to give a 4th axis can be purchased for approximately $150,000 at present. Factoring by 5/4 to give equality of necessary machining degrees of freedom yields:

Cost of conventional 5 axis milling machine $187,500

Thus, it seems the Stewart Platform, as described in Chapter 2 may be a competitor to the conventional milling machine on a cost basis alone.

ACKNOWLEDGEMENTS

The guidance of my supervisors is acknowledged. Dr G. Dunlop for guidance in the development of the control system, and Professor H. McCallion for guidance in the development of the path control method.

Jill Shelton did much fine work related to the presentation of this thesis and her work is greatly appreciated.

The help of the Applied Mechanics Laboratory staff, Garrick Johnson and Norman Bolland for mechanical and electronic aspects respectively is acknowledged.

The support of my parents, June and Bernard, is deeply appreciated.

REFERENCES

1. PHAM, D. T., Techniques, Hardware and Software for Robotic Assembly. Ph. D. Thesis 1979, University of Canterbury, New Zealand.

2. STEWART, D. A., Platform with 6 Degrees of Freedom. Proc. I. Mech. E. (London) 1965-1966, Vol. 180 Pt. 1, No. 15.

3. PRESSMAN, R. and WILLIAMS, J., Numerical Control and Computer Aided Manufacturing. John Wiley & Sons, 1977.

4. SHAH, R., NC Guide Handbook, Volumes 1 and 2. NCA Verlag, Zurich 1979.

5. ACARNELY, P. P., Stepping Motors: A Guide to Modern Theory and Practice. IEE Control Engineering Series 19, 1984.

6. DUFFY, JOSEPH, Analysis of Mechanisms and Robot Manipulators. Edward Arnold, 1980.

7. DUNLOP, G. R., Multiaxis Step Motor Control. [Accepted for publication in the Journal of Instrumentation, Measurement and Control, 1986].

8. ADVANCED MICRO DEVICES. AM9500 Peripheral Products Interface Guide. 901 Thomson Place, P.O. Box 453, Sunnyvale, California.

9. APPARET INC. NEWDOS/80 for the TRS-80. 4401 South Tamarac Parkway, Denver, Colarado.

10. MICROSOFT. TRS-80 Software Package, 1979.

11. YANG, D. C. H., LEE, T. W., Feasibility Study of a Platform Type of Robotic Manipulator from a Kinematic Viewpoint. ASME Journal of Mechanisms and Transmissions etc, Vo. 106, June 1984, Paper 83-DET-35.

12. DEUTSCHMAN, A. D., et al. Machine Design. Collier Macmillan, 1975.

BIBLIOGRAPHY

HUNT, K. H., Kinematic Geometry of Mechanisms. Oxford Engineering Science Series, Clarendon Press, Oxford.

PHAM, D. T. and McCALLION, H., Measuring Errors in a Placement Task. Industrial Robot.

APPENDIX 1

SPECIAL FUNCTIONS

(a) FUNCTION1 creates a 3 x 3 orientation matrix operator [R], called the Rodriguez matrix from the Euler angles ϕ, θ and ψ, i.e.

[R] = FUNCTION1 (ϕ, θ, ψ)

$$= \begin{bmatrix} \cos\phi\cos\theta\cos\psi - \sin\phi\sin\psi & -\cos\phi\cos\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\theta \\ \sin\phi\cos\theta\cos\psi + \cos\phi\sin & -\sin\phi\cos\theta\sin\psi + \cos\phi\cos & \sin\phi\sin\theta \\ -\sin\theta\cos\psi & \sin\theta\sin\psi & \cos\theta \end{bmatrix}$$

(b) FUNCTION2 creates a 3 x 3 orientation matrix operator [R], called the Euler matrix from the direction cosines l, m, n of a line and a rotation about that line of α, i.e.

[R] = FUNCTION2 (l, m, n, α)

$$= \begin{bmatrix} l^2+(1-l^2)\cos\alpha & lm(1-\cos\alpha) - n\sin\alpha & ln(1-\cos\alpha) + m\sin\alpha \\ lm(1-\cos\alpha) + n\sin\alpha & m^2+(1-m^2)\cos\alpha & mn(1-\cos\alpha) - l\sin\alpha \\ ln(1-\cos\alpha) - m\sin\alpha & mn(1-\cos\alpha) + l\sin\alpha & n^2+(1-n^2)\cos\alpha \end{bmatrix}$$

```
00010.C========== COMPILE ========= COMPILE =======
00020.C
00030.C 'COMPILE' COMPILES A SOURCE FILE OF MOVEMENT DATA
00040.C INTO A RUN FILE OF 'MACHINE CONTROL DATA BLOCKS'
00050.C
00060.C    GEOFF RATHBUN          8 / 7 / 85
00070.C
00080.C
00090.         INTEGER BLOCK,LIMIT,MACODE(3500),SPD(100,2),
00100.      X  OFFSET
00110.         LOGICAL MODE,END,FIRST,ERROR,START
00120.         REAL CS(6),CF(6),V,RESN,P(3)
00130.         COMMON /DATA4/SPD
00140.C====== MENU ================
00150.10       WRITE(5,1)
00160.1        FORMAT(1X,50('=')/10X,'COMPILE'/
00170.      X  5X,'1 - COMPILE'/5X,'2 - EXIT TO DOS'/
00180.      X  50('=')/)
00190.         READ(5,2)MODE
00200.2        FORMAT(I1)
00210.         IF(MODE.EQ.1)GO TO 20
00220.         IF(MODE.EQ.2)CALL DOS
00230.         GO TO 10
00240.C================ COMPILE ======================
00250.20       END=.FALSE.
00260.         START=.TRUE.
00270.         FIRST=.TRUE.
00280.         ERROR=.FALSE.
00290.         OFFSET=0
00300.         BLOCK=0
00310.30       CALL INPUT(FIRST,CS,CF,V,RESN,P,END,ERROR)
00320.         IF(ERROR.EQ..TRUE.)GO TO 10
00330.         IF(END.EQ..TRUE.)GO TO 40
00340.         CALL GEOMET(FIRST,CS,CF,V,RESN,P,ERROR,
00350.      X  BLOCK,MACODE,OFFSET,START,END)
00360.         IF(END.EQ..TRUE.)GO TO 40
00370.         IF(ERROR.EQ..TRUE.)GO TO 10
00380.         GO TO 30
00390.C SMOOTH ACCELERATIONS AND SAVE LAST ARRAY LOAD
00400.40       CALL SPEED(MACODE,SPD,BLOCK,ERROR,OFFSET,START,
00410.      X  END)
00420.         IF(ERROR.EQ..TRUE.)GO TO 10
00430.         START=.FALSE.
00440.         LIMIT=10*(BLOCK-OFFSET)+1
00450.         MACODE(LIMIT)=Z'FFFF'
00460.         CALL SAVE(MACODE,LIMIT,START,END)
00470.         GO TO 10
00480.         END
00490.C============= GEOMET ============ GEOMET =========
00500.C
00510.C CREATES ACTUATOR STATE VECTORS ( AS(6) ) FROM
00520.C INPUT MOVEMENT INFORMATION UNIT
00530.C
00540.         SUBROUTINE GEOMET(FIRST,CS,CF,V,RESN,P,
```

```
00550.      X  ERROR,BLOCK,MACODE,OFFSET,START,END)
00560.         INTEGER N,K,MACODE(3500),PC(6),BLOCK,PPC(6),
00570.      X  OFFSET
00580.         REAL CS(6),CF(6),P(3),A,B,C,R,PS(3),PF(3),
00590.      X  RS(3,3),RF(3,3),TK(3),RK(3,3),V,TIME,RESN
00600.         LOGICAL FIRST,ERROR,START,END,FIRST2
00610.         K=1
00620.         IF(FIRST.EQ..TRUE.)K=0
00630.         CALL VECTA(CS,CF,V,RESN,P,A,B,C,R,PS,PF,RS,RF,
00640.      X  TIME,N)
00650.20       CALL VECTB(N,K,A,B,C,R,P,PS,PF,TK,RK,RS)
00660.         CALL LENGTH(TK,RK,PC,ERROR)
00670.         IF(ERROR.EQ..TRUE.)RETURN
00680.C IF THIS IS THE FIRST POSITION, PUT STARTING POSITION
00690.C AT START OF 'MACODE' DATA AND INITIALISE 'PPC'
00700.         IF(FIRST.EQ..FALSE.)GO TO 50
00710.         DO 60 I=1,6
00720.60       PPC(I)=PC(I)
00730.         CALL SAVE(PC,I,START,END)
00740.         FIRST=.FALSE.
00750.         FIRST2=.TRUE.
00760.         K=1
00770.         GO TO 20
00780.C THIS IS NOT THE FIRST POSITION. CREATE 'MCU' CODE
00790.50       CALL KODE(PC,PPC,TIME,BLOCK,MACODE,OFFSET,
00800.      X  START,END,FIRST2,ERROR)
00810.         FIRST2=.FALSE.
00820.         IF(END.EQ..TRUE.)RETURN
00830.         IF(ERROR.EQ..TRUE.)RETURN
00840.         IF(K.EQ.N)RETURN
00850.         K=K+1
00860.         GO TO 20
00870.         END
```

```
00010.C ======== LENGTH ========= LENGTH =========
00020.C
00030.C CALCULATES LEG LENGTHS AND THEN ACTUATOR STATES
00040.C FOR A GIVEN PLATFORM POSITION AND ORIENTATION
00050.C
00060.C DEVELOPED BY PHAM D TRUONG 1978
00070.C MODIFIED  BY GEOFF RATHBUN 1985
00072.C REFER P.H.D. THESIS OF P.D. TRUONG FOR DERIVATION.
00080.C
00090.        SUBROUTINE LENGTH(TK,RK,PC,ERROR)
00100.        LOGICAL ERROR
00110.        REAL TK(3),RK(3,3),LK(6),EBX(3),EBY(3),
00120.     X  EBZ(3),EMVX(3),EMVY(3),EMVZ(3),AX(3),AY(3),
00130.     X  BAX(3),BAY(3),XLEG(6),YLEG(6),ZLEG(6),
00140.     X  EPX(3),EPY(3),BX(3),BY(3),RATIO1,DOTMN,DOTBAN,
00150.     X  XM,YM,ZM,DX,DY,DZ,CONST,WX,WY,WZ,DOTMW,DOTBAW,
00160.     X  RATIO2,ALPHA,UAX,UAY,UAZ,CX,CY,CZ,ILNK,NOT,
00170.     X  JLNK,PITCH,MIN(6),MAX(6)
00180.        INTEGER PC(6)
00190.        COMMON /PHYDAT/PITCH,NOT,JLNK,ILNK,EPX,EPY,
00200.     X  AX,AY,BX,BY,BAX,BAY,MAX,MIN
00210.C CALCULATE 'EB',THE POSITION VECTOR OF THE
00220.C PLATFORM CORNER
00230.        DO 10 I=1,3
00240.        EBX(I)=RK(1,1)*EPX(I)+RK(1,2)*EPY(I)+TK(1)
00250.        EBY(I)=RK(2,1)*EPX(I)+RK(2,2)*EPY(I)+TK(2)
00260.        EBZ(I)=RK(3,1)*EPX(I)+RK(3,2)*EPY(I)+TK(3)
00270.C CALCULATE 'EMV' ,EQUAL TO 'EB-(V*NORMAL)'.
00280.C 'V' IS THE LENGTH OF THE LINK 'DE' AND IS
00290.C REPRESENTED BY 'ILNK'.
00300.        EMVX(I)=EBX(I)-ILNK*RK(1,3)
00310.        EMVY(I)=EBY(I)-ILNK*RK(2,3)
00320.        EMVZ(I)=EBZ(I)-ILNK*RK(3,3)
00330.10      CONTINUE
00340.C CALCULATE 'EMV-A'.
00350.        DO 20 I=1,3
00360.        XM=EMVX(I)-AX(I)
00370.        YM=EMVY(I)-AY(I)
00380.        ZM=EMVZ(I)
00390.C CALCULATE DOT PRODUCT WITH NORMAL.
00400.        DOTMN=XM*RK(1,3)+YM*RK(2,3)+ZM*RK(3,3)
00410.C CALCULATE '(B-A)' DOT NORMAL.
00420.        DOTBAN=BAX(I)*RK(1,3)+BAY(I)*RK(2,3)
00430.C CALCULATE 'W'.
00440.        IF(ABS(DOTBAN).LT.0.0001)GO TO 30
00450.        RATIO1=DOTMN/DOTBAN
00460.        DX=XM-RATIO1*BAX(I)
00470.        DY=YM-RATIO1*BAY(I)
00480.        DZ=ZM
00490.        CONST=1./SQRT(DX*DX+DY*DY+DZ*DZ)
00500.        WX=CONST*DX
00510.        WY=CONST*DY
00520.        WZ=CONST*DZ
00530.        GO TO 40
00540.30      WX=BAX(I)/NOT
00550.        WY=BAY(I)/NOT
```

```
00560.          WZ=0.
00570.C CALCULATE 'U'
00580.40        DOTMW=XM*WX+YM*WY+ZM*WZ
00590.          DOTBAW=BAX(I)*WX+BAY(I)*WY
00600.          RATIO2=DOTMW/DOTBAW
00610.          UAX=XM-RATIO2*BAX(I)
00620.          UAY=YM-RATIO2*BAY(I)
00630.          UAZ=ZM
00640.          ALPHA=JLNK/SQRT(UAX*UAX+UAY*UAY+UAZ*UAZ)
00650.          UX=ALPHA*UAX
00660.          UY=ALPHA*UAY
00670.          UZ=ALPHA*UAZ
00680.C CALCULATE 'C'.
00690.          CX=EMVX(I)-UX
00700.          CY=EMVY(I)-UY
00710.          CZ=EMVZ(I)-UZ
00720.C CALCULATE LEG LENGTHS IN PAIRS.
00730.          K=2*I
00740.          J=K-1
00750.          XLEG(J)=CX-AX(I)
00760.          YLEG(J)=CY-AY(I)
00770.          ZLEG(J)=CZ
00780.          XLEG(K)=CX-BX(I)
00790.          YLEG(K)=CY-BY(I)
00800.          ZLEG(K)=CZ
00810.          LK(J)=SQRT(XLEG(J)*XLEG(J)+YLEG(J)*YLEG(J)+
00820.       X  ZLEG(J)*ZLEG(J))
00830.C CHECK THAT LEG LENGTHS ARE WITHIN LIMITS
00840.          IF(LK(J).GT.MIN(J))GO TO 200
00850.          WRITE(5,11)J
00860.11        FORMAT(' LEG',I2,' MINIMUM LENGTH PASSED')
00870.          GO TO 300
00880.200       IF(LK(J).LT.MAX(J))GO TO 210
00890.          WRITE(5,12)J
00900.12        FORMAT(' LEG',I2,' MAXIMUM LENGTH PASSED')
00910.          GO TO 300
00920.C CALCULATE ACTUATOR STATE OF LEG J
00930.210       PC(J)=INT(LK(J)*PITCH-21000.0)
00940.          LK(K)=SQRT(XLEG(K)*XLEG(K)+YLEG(K)*YLEG(K)+
00950.       X  ZLEG(K)*ZLEG(K))
00960.C CHECK THAT LEG LENGTHS ARE WITHIN LIMITS
00970.          IF(LK(K).GT.MIN(K))GO TO 220
00980.          WRITE(5,11)K
00990.          GO TO 300
01000.220       IF(LK(K).LT.MAX(K))GO TO 230
01010.          WRITE(5,12)K
01020.          GO TO 300
01030.C CALCULATE ACTUATOR STATE OF LEG K
01040.230       PC(K)=INT(LK(K)*PITCH-21000.0)
01050.20        CONTINUE
01060.          RETURN
01070.300       ERROR=.TRUE.
01080.          RETURN
01090.          END
01100.C ======= PHYSICAL ======== PHYSICAL ===============
01110.C
```

```
01120.C PROGRAM INITIALISES TABLE PARAMETERS
01130.C
01140.          BLOCK DATA PHYSIC
01150.          REAL PITCH,EPX(3),
01160.       X  EPY(3),AX(3),AY(3),BX(3),BY(3),BAX(3),BAY(3),
01170.       X  NOP,NOT,ILNK,JLNK,MAX(6),MIN(6)
01180.          COMMON /PHYDAT/PITCH,NOT,JLNK,ILNK,EPX,EPY,
01190.       X  AX,AY,BX,BY,BAX,BAY,MAX,MIN
01200.          DATA PITCH,NOT/78.74,310./,
01210.       X  ILNK,JLNK/15.,7./,
01220.       X  AX/-155.,211.5,-56.5/,
01230.       X  AY/-154.73,-56.87,211.6/,
01240.       X  EPX/0.,131.5,-131.5/,
01250.       X  EPY/-151.84,75.92,75.92/,
01260.       X  BX/155.0,56.5,-211.5/,
01270.       X  BY/-154.73,211.6,-56.87/,
01280.       X  BAX/310.,-155.,-155./,
01290.       X  BAY/0.,268.47,-268.47/,
01300.       X  MAX/415.,415.,415.,415.,395.,432./,
01310.       X  MIN/283.8,271.,287.,270.,267.,287.7/
01320.          END
01330.C ========== VECTA ================= VECTA ==========
01340.C
01350.C CALCULATES TRANSFORMATION DATA FOR ONE MOVE
01360.C FROM 'CS' TO 'CF'. ( SIX SPACE COORDINATES )
01370.C
01380.          SUBROUTINE VECTA(CS,CF,V,RESN,P,A,B,C,R,
01390.       X  PS,PF,RS,RF,TIME,N)
01400.          REAL PS(3),PF(3),CS(6),CF(6),RS(3,3),
01410.       X  RF(3,3),RSF(3,3),A,B,C,R,KOS,KOSQ,TIME,V,MOVE,
01420.       X  M1,M2,M3,R1(6),R2(6),RESN
01430.          INTEGER N
01440.          DO 200 I=1,3
01450.          PS(I)=CS(I)
01460.200       PF(I)=CF(I)
01470.          DO 100 I=4,6
01480.          R1(I)=CS(I)*0.0174533
01490.100       R2(I)=CF(I)*0.0174533
01500.          CALL RMAT(R1(4),R1(5),R1(6),RS)
01510.          CALL RMAT(R2(4),R2(5),R2(6),RF)
01520.          DO 300 I=1,3
01530.          DO 300 J=1,3
01540.300       RSF(I,J)=RF(I,1)*RS(J,1)+RF(I,2)*RS(J,2)
01550.       X  +RF(I,3)*RS(J,3)
01560.C CALCULATE CONSTANTS FROM RSF NEEDED TO FORM RSFK
01570.          KOS=(RSF(1,1)+RSF(2,2)+RSF(3,3)-1.)*0.5
01580.          KOSQ=KOS*KOS
01590.          R=ATAN(SQRT(ABS((1.0-KOSQ)/KOSQ)))
01600.          CHECK=ABS(R)
01610.          IF(CHECK.LT.0.000001)GO TO 400
01620.          CHECK=ABS(R-3.1415926)
01630.          IF(CHECK.LT.0.000001)GO TO 60
01640.          S=SIN(R)*2.0
01650.          A=(RSF(3,2)-RSF(2,3))/S
01660.          B=(RSF(1,3)-RSF(3,1))/S
01670.          C=(RSF(2,1)-RSF(1,2))/S
```

```
01680.          GO TO 450
01690.60        R=3.1415926
01700.          A=SQRT((RSF(1,1)+1.)*0.5)
01710.          B=SQRT((RSF(2,2)+1.)*0.5)
01720.          C=SQRT((RSF(3,3)+1.)*0.5)
01730.          GO TO 450
01740.400       R=0.0
01750.          A=1.0
01760.          B=1.0
01770.          C=1.0
01780.C CALCULATE LINEAR MOVEMENT FROM CS TO CF
01790.450       M1=PF(1)-PS(1)
01800.          M2=PF(2)-PS(2)
01810.          M3=PF(3)-PS(3)
01820.          MOVE=SQRT(M1*M1+M2*M2+M3*M3)+R*130.0
01830.C CALCULATE NUMBER OF MOVES FROM CS TO CF
01840.C USING RESOLUTION VALUE
01850.20        N=INT(MOVE/SQRT(150.0*RESN))
01860.          IF(N.EQ.0)N=1
01870.C CALCULATE TIME FOR PATH VECTOR MOVEMENT
01880.          TIME=MOVE/(N*V)
01890.          RETURN
01900.          END
01910.C============= VECTB ============= VECTB ========
01920.C
01930.C CALCULATES TK , RK FOR THE K'TH POSITION BETWEEN
01940.C CS AND CF.THIS DATA IS FED TO 'LENGTH'.
01950.C
01960.          SUBROUTINE VECTB(N,K,A,B,C,R,P,PS,PF,TK,RK,RS)
01970.          REAL RK(3,3),RS(3,3),RSFK(3,3),PF(3),PS(3),
01980.      X   P(3),RKP(3),PK(3),TK(3),A,B,C,R
01990.          INTEGER K,N
02000.          RATIO=FLOAT(K)/N
02010.          T=RATIO*R
02020.          IF(K.NE.0) GO TO 400
02030.          DO 370 I=1,3
02040.          DO 370 J=1,3
02050.370       RK(I,J)=RS(I,J)
02060.          GO TO 420
02070.400       CALL EMAT(A,B,C,T,RSFK)
02080.          DO 410 I=1,3
02090.          DO 410 J=1,3
02100.410       RK(I,J)=RSFK(I,1)*RS(1,J)+RSFK(I,2)*RS(2,J)
02110.      X   +RSFK(I,3)*RS(3,J)
02120.420       CONTINUE
02130.          DO 470 I=1,3
02140.470       RKP(I)=RK(I,1)*P(1)+RK(I,2)*P(2)+
02150.      X   RK(I,3)*P(3)
02160.          DO 430 I=1,3
02170.430       PK(I)=PS(I)+RATIO*(PF(I)-PS(I))
02180.          DO 500 I=1,3
02190.500       TK(I)=PK(I)-RKP(I)
02200.65        RETURN
02210.          END
02220.C====== EMAT ======== EMAT ======== EMAT =========
02230.C
02240.C CALCULATES EULER MATRIX (3 BY 3)
```

```
02250.C
02260.        SUBROUTINE EMAT(A,B,C,T,RSFK)
02270.        DIMENSION RSFK(3,3)
02280.        D=SIN(T)
02290.        E=COS(T)
02300.        RSFK(1,1)=A*A+(1.-A*A)*E
02310.        RSFK(1,2)=A*B*(1.-E)-C*D
02320.        RSFK(1,3)=A*C*(1.-E)+B*D
02330.        RSFK(2,1)=A*B*(1.-E)+C*D
02340.        RSFK(2,2)=b*B+(1.-B*B)*E
02350.        RSFK(2,3)=B*C*(1.-E)-A*D
02360.        RSFK(3,1)=A*C*(1.-E)-B*D
02370.        RSFK(3,2)=B*C*(1.-E)+A*D
02380.        RSFK(3,3)=C*C+(1.-C*C)*E
02390.        RETURN
02400.        END
02410.C======== RMAT ====== RMAT ====== RMAT =======
02420.C
02430.C CALCULATES RODRIGUEZ MATRIX (3 BY 3)
02440.C
02450.        SUBROUTINE RMAT(F,T,S,R)
02460.        DIMENSION R(3,3)
02470.        CF=COS(F)
02480.        SF=SIN(F)
02490.        CT=COS(T)
02500.        ST=SIN(T)
02510.        CS=COS(S)
02520.        SS=SIN(S)
02530.        R(1,1)=CF*CT*CS-SF*SS
02540.        R(1,2)=-CF*CT*SS-SF*CS
02550.        R(1,3)=CF*ST
02560.        R(2,1)=SF*CT*CS+CF*SS
02570.        R(2,2)=-SF*CT*SS+CF*CS
02580.        R(2,3)=SF*ST
02590.        R(3,1)=-ST*CS
02600.        R(3,2)=ST*SS
02610.        R(3,3)=CT
02620.        RETURN
02630.        END
```

```
00010.C========== KODE ========== KODE ==========
00020.C
00030.C CALCULATES:
00040.C           - PATH VECTOR FROM ASP TO ASC
00050.C           - NUMBER OF SYSTEM VECTORS NEEDED (N)
00060.C TO SPAN THE PATH VECTOR
00070.C           - MACHINE CONTROL DATA BLOCK (MCDB) FOR
00080.C EACH SYSTEM VECTOR
00090.C
00100.C****  SET TO GENERATE '50 STEP' SYSTEM-VECTORS  ****
00110.C
00120.C MAY 'SPEED PROCESS AND SAVE' DATA IF 'MACODE' ARRAY
00130.C IS ALMOST FULL
00140.C
00150.C RETURNS TO TERMINAL 'SPEED PROCESS AND SAVE' IF
00160.C TOTAL NUMBER OF BLOCKS EXCEEDS 1100 ( CCS LIMIT )
00170.C
00180.         SUBROUTINE KODE(ASC,ASP,TIME,BLOCK,MACODE,
00190.      X  OFFSET,START,END,FIRST2,ERROR)
00200.         INTEGER ASC(6),ASP(6),M(6),S(6),D,MAX,FAST,
00210.      X  MN(6),BLOCK,DD,INTMSK,RSD,MACODE(3500),
00220.      X  MOVE(6),B(8),OFFSET,SPD(100,2)
00230.         LOGICAL START,END,FIRST2,ERROR
00240.         REAL MOVERN(6),MNR,ROUND,NFDF,TIME,VEL,SYSVEC
00250.         COMMON /DATA4/SPD
00260.         DATA B,F/1,2,4,8,16,32,64,128,4915200.0/,
00270.      X  SYSVEC/50./
00280.         D=0
00290.         MAX=0
00300.C CALCULATE INITIAL MOVEMENT DATA
00310.         DO 20 I=1,6
00320.         MOVE(I)=0
00330.         M(I)=ASC(I)-ASP(I)
00340.         S(I)=ISIGN(1,M(I))
00350.         M(I)=IABS(M(I))
00360.         IF(S(I).LT.0)GO TO 10
00370.         D=D+B(I)
00380.10       ASP(I)=ASC(I)
00390.         IF(M(I).LT.MAX)GO TO 20
00400.         MAX=M(I)
00410.         FAST=I
00420.20       CONTINUE
00430.C CALCULATE SPEED AND NUMBER OF SYSTEM VECTORS 'N'
00440.30       VEL=FLOAT(MAX)/TIME
00450.         IF(MAX.GT.SYSVEC)GO TO 40
00460.         N=1
00470.         GO TO 45
00480.40       N=INT(FLOAT(MAX)/SYSVEC)+1
00490.45       DO 50 J=1,6
00500.50       MOVERN(J)=FLOAT(M(J))/N
00510.         INTMSK=255-B(FAST)
00520.C CALCULATE REMAINING DATA FOR EACH N
00530.         DO 140 I=1,N
00540.         MAX=0
00550.         DO 60 J=1,6
00560.         IF(I.NE.N)GO TO 55
```

```
00570.          MN(J)=M(J)-MOVE(J)
00580.          GO TO 70
00590.55        MNR=I*MOVERN(J)-MOVE(J)
00600.          IMNR=INT(MNR)
00610.          ROUND=MNR-IMNR
00620.          IF(ROUND.LT.0.5)GO TO 65
00630.          MN(J)=IMNR+1
00640.          GO TO 70
00650.65        MN(J)=IMNR
00660.70        IF(MN(J).LT.MAX)GO TO 60
00670.          MAX=MN(J)
00680.          FAST=J
00690.60        MOVE(J)=MOVE(J)+MN(J)
00700.C OUTPUT AT THIS STAGE:
00710.C         MN(J)    J=1 TO 6 FOR I=1 TO N
00720.C         FAST     FASTEST MOTOR FOR EVERY I
00730.C         D        DIRECTION WORD.SAME FOR ALL I
00740.C         MAX      MAXIMUM MOVE FOR EVERY I
00750.C
00760.          IF(I.NE.1)GO TO 300
00770.C
00780.C--------- FIRST LOOP OF 'KODE' -----------
00790.C
00800.C CALCULATE SPEED SMOOTHING DATA AND IF NECESSARY
00810.C SMOOTH AND SAVE MACODE DATA IF MACODE IS FULL.
00820.C IF # BLOCKS ¶ 1100 ( UPPER LIMIT ) SAVE REMAINDER
00830.C AND EXIT
00840.          IF((BLOCK+N).LT.1100)GO TO 100
00850.C SAVE REMAINING DATA
00860.          END=.TRUE.
00870.          WRITE(5,2)
00880.2         FORMAT(' DATA OVERFLOW. SAVING EXISTING DATA')
00890.          RETURN
00900.100       CALL SPDDAT(BLOCK,MN,S,FAST,OFFSET,FIRST2,END)
00910.          SIZE=BLOCK-OFFSET+N
00920.C MACODE IS FULL
00930.          IF(SIZE.LT.350)GO TO 300
00940.          LIMIT=10*(BLOCK-OFFSET)
00950.          CALL SPEED(MACODE,SPD,BLOCK,ERROR,OFFSET,START,
00960.       X  END)
00970.          IF(ERROR.EQ..TRUE.)RETURN
00980.          START=.FALSE.
00990.          CALL SAVE(MACODE,LIMIT,START,END)
01000.          OFFSET=BLOCK
01010.C CONTINUE WITH NEW OFFSET
01020.C
01030.C NOW CALCULATE AND LOAD MACHINE CODE BLOCKS
01040.C INTO 'MACODE'
01050.C
01060.C-----------------------------------------
01070.300       L=10*(BLOCK-OFFSET)+1
01080.C LOAD BLOCK NUMBER...-- 1 --
01090.          MACODE(L)=BLOCK
01100.C CALCULATE 'DIRG'
01110.          ABLOCK=(BLOCK+2)*0.5
```

```
01120.          BBLOCK=INT(ABLOCK)
01130.          DD=D
01140.          IF(ABLOCK.EQ.BBLOCK)GO TO 90
01150.          DD=D+192
01160.90        VALUE=256.0*FLOAT(DD)+INTMSK
01170.          IF(VALUE.GT.32767.0)VALUE=-1*(65536.-VALUE)
01180.C LOAD INTMSK AND DIRG WORDS..-- 2 --
01190.          MACODE(L+1)=INT(VALUE)
01200.C LOAD TARGET STEP COUNT..-- 3 --
01210.          MACODE(L+2)=MAX
01220.          NFDF=491.0*MAX
01230.C CALCULATE AND LOAD RSD(6). CALCULATE THE STOP BYTE
01240.          DO 110 K=1,6
01250.          IF(MN(K).NE.0)GO TO 95
01260.C PLACING FFFEH INTO DIRECTION DIVISOR EFFECTIVELY
01270.C 'STOPS' THAT MOTOR OVER TIME INTERVAL
01280.          RSD=Z'FFFE'
01290.          GO TO 120
01300.95        VALUE=NFDF/FLOAT(MN(K))
01310.          IF(VALUE.GT.32767.)VALUE=-1.0*(65536.0-VALUE)
01320.          RSD=INT(VALUE)
01330.120       IM=L+K+2
01340.C LOAD RELATIVE SPEED DIVISORS          -- 4 TO 9 --
01350.110       MACODE(IM)=RSD
01360.C LOAD SPEED DIVISOR                    -- 10 --
01370.          K=INT(10000./VEL)
01380.          MACODE(L+9)=K
01390.          BLOCK=BLOCK+1
01400.140       CONTINUE
01410.          WRITE(5,5)BLOCK
01420.5         FORMAT(I5)
01430.          RETURN
01440.          END
```

```
00010.C ========== INPUT ========== INPUT =======
00020.C
00030.C EXTRACTS SUCCESIVE MIU'S OUT OF A SOURCE FILE
00040.C AN MIU CONTAINS : CS - STARTING COORDINATE IN SS
00050.C                 : V  - FEED SPEED
00060.C                 : R  - PATH RESOLUTION
00070.C                 : P  - PLATFORM 'ZERO' VECTOR
00080.C                 : CF - FINISHING COORDINATE IN SS
00090.C
00100.        SUBROUTINE INPUT(FIRST,CS,CF,V,RESN,P,ENDI,
00110.     X  ERROR)
00120.        REAL CS(6),CF(6),V,RESN,P(3)
00130.        LOGICAL SOURCE(4000),BUFFER(64),FIRST,ENDI,
00140.     X  ERROR
00150.        INTEGER S
00160.        COMMON /DATA3/SOURCE,BUFFER,S
00170.        IF(FIRST.EQ..FALSE.)GO TO 20
00180.C THIS PATH IS TAKEN ON THE FIRST CALL OF 'INPUT'
00190.C VALUES OF CS(6),V,R,P(3) AND CF(6) ARE EXPECTED
00200.        S=15
00210.        CALL LOAD(SOURCE)
00220.        CALL BUFF
00230.        DECODE(BUFFER,1)(CS(I),I=1,6)
00240.1       FORMAT(6F7.2)
00250.        IF(SOURCE(S).NE.Z'56')GO TO 70
00260.        CALL BUFF
00270.        DECODE(BUFFER,2)V
00280.2       FORMAT(F7.2)
00290.        IF(SOURCE(S).NE.Z'52')GO TO 70
00300.        CALL BUFF
00310.        DECODE(BUFFER,2)RESN
00320.        IF(SOURCE(S).NE.Z'50')GO TO 70
00330.        CALL BUFF
00340.        DECODE(BUFFER,3)(P(I),I=1,3)
00350.3       FORMAT(3F7.2)
00360.        IF(SOURCE(S).NE.Z'43')GO TO 70
00370.        CALL BUFF
00380.        DECODE(BUFFER,1)(CF(I),I=1,6)
00390.        RETURN
00400.C ENTRY POINT FOR ANY ENTRY AFTER FIRST ENTRY
00410.C OPTIONAL INPUT OF V,R AND P(3) VALUES
00420.C THERE MUST BE A NEW CF(6) VALUE OR AN END CODE
00430.20      CONTINUE
00440.        DO 30 I=1,6
00450.30      CS(I)=CF(I)
00460.        IF(SOURCE(S).NE.Z'56')GO TO 40
00470.        CALL BUFF
00480.        DECODE(BUFFER,2)V
00490.40      IF(SOURCE(S).NE.Z'52')GO TO 50
00500.        CALL BUFF
00510.        DECODE(BUFFER,2)RESN
00520.50      IF(SOURCE(S).NE.Z'50')GO TO 60
00530.        CALL BUFF
00540.        DECODE(BUFFER,3)(P(I),I=1,3)
00550.60      IF(SOURCE(S).NE.Z'43')GO TO 90
00560.        CALL BUFF
```

```
00570.          DECODE(BUFFER,1)(CF(I),I=1,6)
00580.          RETURN
00590.90        IF(SOURCE(S).EQ.Z'45'.AND.SOURCE(S+1).EQ.Z'4E')
00600.      X   GO TO 80
00610.70        WRITE(5,7)
00620.7         FORMAT(' FILE ERROR')
00630.          ERROR=.TRUE.
00640.          RETURN
00650.80        ENDI=.TRUE.
00660.          RETURN
00670.          END
00680.C ========= BUFF ===============
00690.C
00700.C FILLS A BUFFER ARRAY WITH CONTENTS OF AN 'EDIT' LINE
00710.C FOR SUBSEQUENT DECODING INTO DATA
00720.C
00730.          SUBROUTINE BUFF
00740.          LOGICAL BUFFER(64),SOURCE(4000)
00750.          INTEGER S
00760.          COMMON /DATA3/SOURCE,BUFFER,S
00770.          DO 10 I=1,50
00780.          S=S+1
00790.          BUFFER(I)=SOURCE(S)
00800.          IF(BUFFER(I).EQ.Z'0D')GO TO 20
00810.10        CONTINUE
00820.20        S=S+15
00830.          RETURN
00840.          END
00850.C ======== LOAD ========= LOAD ===========
00860.C
00870.C LOADS COMPLETE FILE FROM DISC
00880.C
00890.          SUBROUTINE LOAD(ARRAY)
00900.          INTEGER ARRAY(2000)
00910.          REAL*8 H
00920.20        WRITE(5,1)
00930.1         FORMAT(' SOURCE FILENAME ?'/)
00940.          READ(5,2)H
00950.2         FORMAT(A6)
00960.          CALL OPEN(7,H,2)
00970.          DO 10 I=1,2000
00980.          READ(7,ERR=200,END=100)ARRAY(I)
00990.10        CONTINUE
01000.100       ENDFILE 7
01010.          RETURN
01020.200       WRITE(5,4)
01030.4         FORMAT(' I/O ERROR')
01040.          GO TO 20
01050.          END
01060.C ========= SAVE ======= SAVE =======
01070.C
01080.C OPENS A DISC FILE , THEN SAVES ARRAYS PASSED IN
01090.C SUCCESSIVE CALLS UNTIL TOLD TO CLOSE FILE
01100.C BY AN 'END' FLAG
```

```
01110.C
01120.          SUBROUTINE SAVE(ARRAY,LIMIT,START,END)
01130.          INTEGER ARRAY(3500),LIMIT
01140.          LOGICAL START,END
01150.          REAL*8 H
01160.          IF(START.EQ..FALSE.)GO TO 30
01170.10        WRITE(5,1)
01180.1         FORMAT(' RUN FILENAME ?'/)
01190.          READ(5,2)H
01200.2         FORMAT(A6)
01210.          CALL OPEN(7,H,2)
01220.30        DO 20 I=1,LIMIT
01230.20        WRITE(7,ERR=100,END=200)ARRAY(I)
01240.          IF(END.EQ..FALSE.)RETURN
01250.          ENDFILE 7
01260.200       RETURN
01270.100       WRITE(5,3)
01280.3         FORMAT(' I/O ERROR')
01290.          GO TO 10
01300.          END


00010.;========== DOS CALL ================
00020.;
00030.          ENTRY DOS
00040.DOS:      IM      1
00050.          JP      402DH
00060.          RET
00070.          END



00010.C=========== SPEED CONTROL PROGRAMS ===========
00020.C
00030.C THESE PROGRAMS POST-PROCESS AN ARRAY OF MCDB'S TO
00040.C KEEP MOTOR ACCELERATIONS BELOW SET LIMIT FUNCTIONS.
00050.C
00060.C=========== SPEED ========== SPEED ==========
00070.C
00080.C 'SPEED' CONTROLS THE POST-PROCESSING
00090.C
00100.          SUBROUTINE SPEED(ARRAY,SPD,ENDBLK,ERROR,
00110.      X   OFFSET,START,END)
00120.          INTEGER BLOCK,ARRAY(3500),N,ENDBLK,
00130.      X   SPD(100,2),VE,VP,OFFSET
00140.          LOGICAL ERROR,START,END
00150.99        WRITE(5,1)
00160.1         FORMAT(' SMOOTHING ACCELERATIONS')
00170.          IF(START.NE..TRUE.)GO TO 5
00180.          VP=100
00190.          BLOCK=0
00200.5         K=1
00210.10        IF(SPD(K,1).EQ.Z'FFFF')GO TO 20
00220.          N=SPD(K,1)
```

```
00230.          VE=SPD(K,2)
00240.          CALL SPDMAX(N,BLOCK,VP,VE,ARRAY,ERROR,OFFSET,
00250.       X  START,END)
00260.          IF(ERROR.EQ..TRUE.)RETURN
00270.          K=K+1
00280.          GO TO 10
00290.C IF END OF MOVEMENT SMOOTH LAST PATH VECTOR TO
00300.C VE=100
00310.20        IF(END.EQ..TRUE.)GO TO 30
00320.          START=.FALSE.
00330.          RETURN
00340.30        N=ENDBLK-BLOCK
00350.          VE=100
00360.          CALL SPDMAX(N,BLOCK,VP,VE,ARRAY,ERROR,OFFSET,
00370.       X  START,END)
00380.          RETURN
00390.          END
00400.C
00410.C============ SPDMAX ========================
00420.C
00430.C 'SPDMAX' 'FILTERS' OUT WRONG OR EXCESSIVE PARAMETERS
00440.C
00450.          SUBROUTINE SPDMAX(N,BLOCK,VP,VE,ARRAY,ERROR,
00460.       X  OFFSET)
00470.          INTEGER ARRAY(3500),N,BLOCK,VT,VTMAX,VP,VE,
00480.       X  N,NMIN,VEMIN,VTMIN,TYPE,OFFSET
00490.          LOGICAL ERROR
00500.          REAL KA2,KD
00510.          DATA KA2,KD/25000.,100./
00520.          L=10*(BLOCK-OFFSET)+1
00530.C         CHECK IF BLOCK NUMBERS AGREE
00540.          IF(ARRAY(L).EQ.BLOCK)GO TO 10
00550.          WRITE(5,2)
00560.2         FORMAT(' BLOCK # ERROR')
00570.          ERROR=.TRUE.
00580.          RETURN
00590.C GET 'VT'
00600.10        VT=10000./ARRAY(L+9)
00610.          IF(VT.LE.500)GO TO 15
00620.          WRITE(5,9)VT
00630.9         FORMAT(1X,I8,' HELD TO 500')
00640.          VT=500
00650.15        IF(VT.EQ.VP)GO TO 500
00660.          IF(VT.LT.VP)GO TO 200
00670.C ACCELERATION DESIRED
00680.          IF(VT.LE.VE)GO TO 100
00690.C ACCELERATE, THEN DECELERATE.   == A ==
00700.C CALCULATE VT MAX
00710.          C=4.*(FLOAT(VP)*VP/KA2+VE/KD+N-1.)/KA2
00720.          VTMAX=(SQRT(C)-1./KD)*KA2/2.
00730.          IF(VTMAX.LT.VP)GO TO 50
00740.          IF(VT.LE.VTMAX)GO TO 20
00750.          WRITE(5,4)VT,VTMAX,BLOCK
00760.4         FORMAT(1X,I6,' HELD TO',I6,' IN BLOCK',I5)
00770.          VT=VTMAX
```

```
00780.20       CALL SMOOTH(VP,VT,VE,N,BLOCK,ARRAY,ERROR,
00790.      X  OFFSET)
00800.         RETURN
00810.C JUST ACCELERATE.        == D ==
00820.100      VTMAX=SQRT(KA2*(N-1)+FLOAT(VP)*VP)
00830.         IF(VT.LE.VTMAX)GO TO 30
00840.         VT=VTMAX
00850.30       CALL SMOOTH(VP,VT,VE,N,BLOCK,ARRAY,ERROR,
00860.      X  OFFSET)
00870.         RETURN
00880.C DECELERATE
00890.C CASES            == B ==  AND  == C ==
00900.200      NMIN=(VP-VE)/KD+1.
00910.         IF(NMIN.GT.N)GO TO 50
00920.         CALL SMOOTH(VP,VT,VE,N,BLOCK,ARRAY,ERROR,
00930.      X  OFFSET)
00940.         RETURN
00950.C VT=VP
00960.500      IF(VT.GT.VE)GO TO 510
00970.C                                 == F ==
00980.         BLOCK=BLOCK+N
00990.         RETURN
01000.C DECELERATE TO VE FROM VP        == E ==
01010.510      NMIN=(VP-VE)/KD+1.
01020.         IF(NMIN.GT.N)GO TO 50
01030.         CALL SMOOTH(VP,VT,VE,N,BLOCK,ARRAY,ERROR,
01040.      X  OFFSET)
01050.         RETURN
01060.50       WRITE(5,6)
01070.6        FORMAT(' CANNOT ACCELERATE IN # OF BLOCKS')
01080.         ERROR=.TRUE.
01090.         RETURN
01100.         END
01110.C=========== SMOOTH ==================
01120.C
01130.C 'SMOOTH' CONTROLS ALTERATION OF SPEED DIVISORS
01140.C IN THE 'MCDB' ACCORDING TO 'FILTERED' DATA AND TO
01150.C A SET OF ACCELERATION AND DECELERATION FUNCTIONS
01160.C
01170.         SUBROUTINE SMOOTH(VP,VT,VE,N,BLOCK,ARRAY,ERROR,
01180.      X  OFFSET)
01190.         INTEGER VP,VT,VE,N,BLOCK,ARRAY(3500),BA,BB,
01200.      X  OFFSET
01210.         LOGICAL ERROR
01220.         REAL KA2,KA
01230.         DATA KA,KA2/158.11,25000./
01240.C SET UP FIRST BLOCK IN SUB-VECTOR EQUAL TO LAST SPEED
01250.         BA=BLOCK
01260.         CALL SPDLD(BA,VP,ARRAY,OFFSET)
01270.         IF(VT.EQ.VP)GO TO 200
01280.         IF(VT.LT.VP)GO TO 100
01290.C ACCELERATE FROM VP TO VT
01300.C CALCULATE NEXT SPEED
01310.10       A=FLOAT(VP)*VP/KA2+1.0
01320.         VP=KA*SQRT(A)
01330.         IF(VP.LT.VT)GO TO 20
```

```
01340.          VP=VT
01350.          CALL SPDLD(BA,VP,ARRAY,OFFSET)
01360.          GO TO 200
01370.20        CALL SPDLD(BA,VP,ARRAY,OFFSET)
01380.          GO TO 10
01390.C DECELERATE FROM VP TO VT
01400.C CALCULATE NEXT SPEED
01410.100       VP=VP-100
01420.          IF(VP.GT.VT)GO TO 110
01430.          VP=VT
01440.          CALL SPDLD(BA,VP,ARRAY,OFFSET)
01450.          GO TO 200
01460.110       CALL SPDLD(BA,VP,ARRAY,OFFSET)
01470.          GO TO 100
01480.C TERMINAL PHASE. POSSIBLE DECELERATION TO 'VE'
01490.200       IF(VP.GT.VE)GO TO 190
01500.          M=BLOCK+N-BA
01510.          GO TO 300
01520.C CALCULATE DECELERATION STARTING POINT
01530.190       BB=BLOCK+N-FLOAT(VP-VE)/100.
01540.          M=BB-BA
01550.          IF(M.GT.0)GO TO 210
01560.          IF(M.EQ.0)GO TO 240
01570.230       WRITE(5,4)
01580.4         FORMAT(' BLOCK CLASH ON DECELERAION')
01590.          ERROR=.TRUE.
01600.          RETURN
01610.210       VP=VP-100
01620.          IF(VP.LE.VE)GO TO 220
01630.          CALL SPDLD(BB,VP,ARRAY,OFFSET)
01640.          GO TO 210
01650.220       VP=VE
01660.          CALL SPDLD(BB,VP,ARRAY,OFFSET)
01670.C EXIT
01680.C ADJUST SPEED VALUES FROM BA TO BB TO VT
01690.300       DO 60 I=1,M
01700.60        CALL SPDLD(BA,VT,ARRAY,OFFSET)
01710.240       BLOCK=BLOCK+N
01720.          RETURN
01730.          END
01740.C=========== SPDDAT =========== SPDDAT =====
01750.C
01760.C 'SPDDAT' GATHERS DATA NEEDED BY 'SPEED' FROM 'KODE'.
01770.C MINIMUM TRANS NODE SPEED 'VE' IS CALCULATED AND
01780.C NUMBER OF SYSTEM VECORS IN PRECEEDING PATH VECTOR ARE
01790.C PLACED IN 'SPD' ARRAY
01800.C
01810.          SUBROUTINE SPDDAT(BLOCK,MI,S,FAST,OFFSET,FIRST,
01820.       X  END)
01830.          INTEGER S(6),LASTS(6),BLOCK,LSTBLK,MI(6),
01840.       X  SPD(100,2),FAST,VE,VEA,VED,VEDC,OFFSET,
01850.       X  LSTOST,LSTFST
01860.          LOGICAL FIRST,END
01870.          REAL M(6),LASTM(6),A,AMAX,AMIN,KA,KD
01880.          COMMON /DATA4/SPD
```

```
01890.          DATA KA,KD/158.11,100./
01900.          DO 50 I=1,6
01910.50        M(I)=FLOAT(MI(I))
01920.          IF(FIRST.EQ..FALSE.)GO TO 20
01930.          K=1
01940.          LSTOST=0
01950.          GO TO 70
01960.20        IF(OFFSET.EQ.LSTOST)GO TO 30
01970.          K=1
01980.          LSTOST=OFFSET
01990.30        VEDC=500
02000.          VEA=500
02010.          VED=500
02020.          AMAX=0.05
02030.          AMIN=1.
02040.          DO 40 I=1,6
02050.C CALCULATE TRANS NODE ACCELERATION FOR MOTOR 'I'
02060.          A=M(I)/M(FAST)-LASTM(I)/LASTM(LSTFST)
02070.          IF(A.GT.AMAX)AMAX=A
02080.          IF(A.LT.AMIN)AMIN=A
02090.C CALCULATE VEDC IF A DIRECTION CHANGE OCCURS
02100.          IF(S(I).EQ.LASTS(I))GO TO 40
02110.          VE=150./(M(I)/M(FAST)+LASTM(I)/LASTM(LSTFST))
02120.          IF(VE.LT.VEDC)VEDC=VE
02130.40        CONTINUE
02140.C CALCULATE VEA,VED AND THEN GET MINIMUM ENDING SPEED
02150.C 'VE' FROM VEA,VED AND VEDC.
02160.          VEA=KA/SQRT(AMAX*(AMAX+2.))
02170.          IF(AMIN.GE.0.0)GO TO 55
02180.          VED=ABS(KD/AMIN)
02190.55        VE=MIN0(VEA,VED,VEDC)
02200.C SAVE DATA INTO SPEED ARRAY
02210.          SPD(K,1)=BLOCK-LSTBLK
02220.          SPD(K,2)=VE
02230.C PUT 'EOF' CODE INTO NEXT ARRAY ELEMENT
02240.          SPD(K+1,1)=Z'FFFF'
02250.          K=K+1
02260.C SAVE PRESENT DATA FOR NEXT CALL OF THIS PROGRAM
02270.70        LSTBLK=BLOCK
02280.          LSTFST=FAST
02290.          DO 80 I=1,6
02300.          LASTS(I)=S(I)
02310.80        LASTM(I)=M(I)
02320.          RETURN
02330.          END
02340.C========== SPDLD ========== SPDLD ========
02350.C
02360.          SUBROUTINE SPDLD(B,V,ARRAY,OFFSET)
02370.          INTEGER B,V,ARRAY(3500),VP,VC,OFFSET
02380.          L=10*(B-OFFSET)+10
02390.          VC=10000./V
02400.          ARRAY(L)=VC
02410.          B=B+1
02420.          RETURN
02430.          END
```

```
00010.C======== COMPUTER CONTROL SYSTEM (CCS) ========
00020.C
00030.C THESE PROGRAMS PROVIDE FULL CONTROL OVER THE
00040.C UNIVERSITY OF CANTERBURY STEWART PLATFORM.
00050.C
00060.C THEY USE DATA FROM A 'RUN' FILE TO PROVIDE
00070.C MOTION OF THE STEWART PLATFORM.
00080.C
00090.C CREATED BY  GEOFF RATHBUN   9 / 7 / 85
00100.C
00110.C============ MENU ======================
00120.         LOGICAL MODE
00130.10       WRITE(5,1)
00140.1        FORMAT(1X,40('=')/
00150.      X  '  STEWART PLATFORM CONTROL PROGRAM'/
00160.      X  8X,'1 -- RUN A MACHINE CODE FILE'/
00170.      X  8X,'2 -- ZERO THE TABLE'/
00180.      X  8X,'3 -- EXIT TO NEWDOS'/1X,40('=')/)
00190.         READ(5,2)MODE
00200.2        FORMAT(I5)
00210.         IF(MODE.EQ.1)CALL RUN
00220.         IF(MODE.EQ.2)CALL ZERO
00230.         IF(MODE.EQ.3)CALL DOS
00240.         GO TO 10
00250.         END
00260.C============== RUN =============== RUN ==========
00270.C
00280.C CONTROLS RUNNING OF A 'RUN' FILE
00290.C
00300.         SUBROUTINE RUN
00310.         INTEGER MACODE(11000),AS(6),ASZ(6),
00320.      X  ASC(6),ASST(6),BLOCK,SDIV
00330.         LOGICAL NSTOP,A,B,ZERO,START
00340.         REAL TIME
00350.         COMMON /PLACE/AS,ASZ
00360.         BLOCK=0
00370.         MESG=.TRUE.
00380.         START=.TRUE.
00390.         ZERO=.FALSE.
00400.         NSTOP=.FALSE.
00410.         A=NSTOP
00420.         B=.FALSE.
00430.C GET STARTING POSITION 'ASST' FROM DATA FILE
00440.         CALL LOAD(START,ASST)
00450.         START=.FALSE.
00460.10       WRITE(5,2)
00470.2        FORMAT(' STARTING FROM ZERO ? Y/N'/)
00480.         READ(5,6)K
00490.6        FORMAT(A1)
00500.         IF(K.NE.'Y'.AND.K.NE.'N')GO TO 10
00510.         IF(K.EQ.'N')GO TO 20
00520.         DO 50 I=1,6
00530.50       AS(I)=ASZ(I)
00540.20       WRITE(5,4)
00550.4        FORMAT(' STOP AT NODES ? Y/N'/)
00560.60       READ(5,6)K
```

```
00570.          IF(K.NE.'Y'.AND.K.NE.'N')GO TO 60
00580.          IF(K.EQ.'Y')NSTOP=.TRUE.
00590.C CALCULATE MCDB'S FOR MOVEMENT FROM AS TO ASC AND
00600.C PUT THESE DATA BLOCKS INTO THE 'MACODE' ARRAY.
00610.          DO 40 I=1,6
00620.40        ASC(I)=AS(I)
00630.          WRITE(5,1)
00640.1         FORMAT(' SPEED TO START POINT ? STEPS/SEC'/)
00650.          READ(5,3)SDIV
00660.3         FORMAT(I3)
00670.          SDIV=INT(10000./FLOAT(SDIV))
00680.          TIME=1.0
00690.          CALL KODE(ASST,ASC,TIME,BLOCK,MACODE,SDIV)
00700.          LIM=10*BLOCK+1
00710.          MACODE(LIM)=Z'FFFF'
00720.C GO TO STARTING POSITION
00730.          CALL SETUP
00740.65        WRITE(5,9)
00750.9         FORMAT(' READY. G - GO , X - EXIT'/)
00760.          READ(5,6)K
00770.          IF(K.NE.'G'.AND.K.NE.'X')GO TO 65
00780.          IF(K.EQ.'X')RETURN
00790.          CALL GO(MACODE(1),A,B,AS(1))
00800.          WRITE(5,8)
00810.8         FORMAT(' STARTING POSITION ACHIEVED.
00820.     X    LOADING REST OF FILE.')
00830.C LOAD MACODE ARRAY WITH REMAINDER OF MOVEMENT FILE
00840.          CALL LOAD(START,MACODE)
00850.55        WRITE(5,9)
00860.          READ(5,6)K
00870.          IF(K.NE.'G'.AND.K.NE.'X')GO TO 55
00880.          IF(K.EQ.'X')RETURN
00890.C NOW EXECUTE COMMANDED MOVEMENT
00900.          CALL GO(MACODE(7),NSTOP,ZERO,AS(1))
00910.C MOVEMENT COMPLETED
00920.          WRITE(2,12)
00930.12        FORMAT(1X,50('-')/)
00940.          RETURN
00950.          END
00960.C========== ZERO ========== ZERO ======
00970.C
00980.C MOVES PLATFORM TO THE 'ZERO' POSITION AND INITIALISES
00990.C AS TO ASZ
01000.C
01010.          SUBROUTINE ZERO
01020.          INTEGER B,K,MACODE(21),AS(6),ASZ(6),SPEED
01030.          LOGICAL NSTOP,ZEROO
01040.          COMMON /PLACE/AS,ASZ
01050.          DATA ASZ/1346,339,1598,260,24,1654/
01060.          MACODE(1)=0
01070.          MACODE(2)=255
01080.          MACODE(3)=20000
01090.          WRITE(5,3)
01100.3         FORMAT(' SPEED ? STEPS/SEC'/)
01110.          READ(5,4)SPEED
01120.4         FORMAT(I5)
01130.          SPEED=INT(32000.0/SPEED)
```

```
01140.          MACODE(10)=SPEED
01150.          DO 40 I=4,9
01160.40        MACODE(I)=150
01170.          CALL SETUP
01180.          NSTOP=.FALSE.
01190.          ZEROO=.TRUE.
01200.          CALL GO(MACODE(1),NSTOP,ZEROO,AS(1))
01210.          WRITE(5,2)
01220.2         FORMAT(' PRESS KEYS 1 TO 6 TO STOP ANY
01230.      X   MOTOR'/' PRESS BREAK TO RETURN TO MENU')
01240.50        CALL SCAN(K)
01250.          IF(K.EQ.33)CALL OUT(2,Z'C4')
01260.          IF(K.EQ.34)CALL OUT(2,Z'C8')
01270.          IF(K.EQ.35)CALL OUT(6,Z'C4')
01280.          IF(K.EQ.36)CALL OUT(6,Z'C8')
01290.          IF(K.EQ.37)CALL OUT(10,Z'C4')
01300.          IF(K.EQ.38)CALL OUT(10,Z'C8')
01310.          IF(K.NE.18)GO TO 50
01320.          CALL OUT(2,Z'DF')
01330.          RETURN
01340.          END
01350.C ========== PRESENT POSITION ============
01360.C
01370.C PRINTS THE VALUE OF THE PRESENT ACTUATOR STATE
01380.C VECTOR - 'AS' (6 ELEMENTS)
01390.C
01400.          SUBROUTINE POSN(BLOCK)
01410.          INTEGER AS(6),BLK,BLOCK
01420.          COMMON /PLACE/AS
01430.          BLK=BLOCK
01440.          IF(BLK.GT.1)BLK=BLK-1
01450.          WRITE(2,1)BLK,(AS(I),I=1,6)
01460.1         FORMAT(' BLOCK, AS(6)',7I8)
01470.          RETURN
01480.          END
01490.C ======== LOAD ========= LOAD ==========
01500.C
01510.C OPENS AND EXTRACTS THE FIRST 12 BYTES OF A FILE
01520.C ON THE FIRST CALL
01530.C NEXT CALL LOADS REMAINDER OF FILE
01540.C
01550.          SUBROUTINE LOAD(START,ARRAY)
01560.          INTEGER ARRAY(11000),LIMIT
01570.          LOGICAL START
01580.          REAL*8 H
01590.          LIMIT=11000
01600.          IF(START.EQ..FALSE.)GO TO 50
01610.          LIMIT=6
01620.20        WRITE(5,1)
01630.1         FORMAT(' FILENAME ?'/)
01640.          READ(5,2)H
01650.2         FORMAT(A6)
01660.50        CALL OPEN(7,H,2)
01670.          DO 10 I=1,LIMIT
01680.          READ(7,ERR=200,END=100)ARRAY(I)
01690.10        CONTINUE
01700.100       ENDFILE 7
```

```
01710.          RETURN
01720.200       WRITE(5,4)
01730.4         FORMAT(' I/O ERROR')
01740.          GO TO 20
01750.          END
01760.C========== KODE ========== KODE ==========
01770.C
01780.C 'KODE' IS USED TO CALCULATE MCBD'S FROM AS TO ASST
01790.C
01800.          SUBROUTINE KODE(ASST,ASC,TIME,BLOCK,ARRAY,SDIV)
01810.          INTEGER ASST(6),ASC(6),M(6),S(6),D,MAX,FAST,
01820.      X   MN(6),BLOCK,DD,INTMSK,RSD(6),ARRAY(11000),
01830.      X   MOVE(6),SDIV,B(8)
01840.          REAL MOVERN(6),MNR,ROUND,NFDF,TIME,SPEED
01850.          DATA B,F/1,2,4,8,16,32,64,128,4915200.0/
01860.          WRITE(5,1)
01870.1         FORMAT(' CALCULATING MOVEMENT TO START POINT'/)
01880.          D=0
01890.          MAX=0
01900.C CALCULATE INITIAL MOVEMENT DATA
01910.          DO 20 I=1,6
01920.          MOVE(I)=0
01930.          M(I)=ASST(I)-ASC(I)
01940.          S(I)=ISIGN(1,M(I))
01950.          M(I)=IABS(M(I))
01960.          IF(S(I).LT.0)GO TO 10
01970.          D=D+B(I)
01980.10        ASC(I)=ASST(I)
01990.          IF(M(I).LT.MAX)GO TO 20
02000.          MAX=M(I)
02010.          FAST=I
02020.20        CONTINUE
02030.          IF(MAX.LE.0)RETURN
02040.C CALCULATE SPEED AND NUMBER OF SUB-BLOCKS 'N'
02050.          SPEED=FLOAT(MAX)/TIME
02060.          IF(MAX.GT.50)GO TO 40
02070.          N=1
02080.          GO TO 45
02090.40        N=INT(FLOAT(MAX)/50.)+1
02100.45        DO 50 J=1,6
02110.50        MOVERN(J)=FLOAT(M(J))/N
02120.          INTMSK=255-B(FAST)
02130.C CALCULATE REMAINING DATA FOR EACH N
02140.          DO 130 I=1,N
02150.          MAX=0
02160.          DO 60 J=1,6
02170.          IF(I.NE.N)GO TO 55
02180.          MN(J)=M(J)-MOVE(J)
02190.          GO TO 70
02200.55        MNR=I*MOVERN(J)-MOVE(J)
02210.          IMNR=INT(MNR)
02220.          ROUND=MNR-IMNR
02230.          IF(ROUND.LT.0.5)GO TO 65
02240.          MN(J)=IMNR+1
02250.          GO TO 70
02260.65        MN(J)=IMNR
02270.70        IF(MN(J).LT.MAX)GO TO 60
02280.          MAX=MN(J)
```

```
02290.          FAST=J
02300.60        MOVE(J)=MOVE(J)+MN(J)
02310.C OUTPUT AT THIS STAGE:
02320.C         MN(J)    J=1 TO 6 FOR I=1 TO N
02330.C         FAST     FASTEST MOTOR FOR EVERY I
02340.C         D        DIRECTION WORD.SAME FOR ALL I
02350.C         MAX      MAXIMUM MOVE FOR EVERY I
02360.C
02370.C NOW CALCULATE AND LOAD MACHINE CODE BLOCKS
02380.C INTO 'ARRAY' ARRAY
02390.          L=10*BLOCK+1
02400.C LOAD BLOCK NUMBER.    .-- 1 --
02410.          ARRAY(L)=BLOCK
02420.          ABLOCK=(BLOCK+2)*0.5
02430.          BBLOCK=INT(ABLOCK)
02440.          DD=D
02450.          IF(ABLOCK.EQ.BBLOCK)GO TO 90
02460.          DD=D+192
02470.90        VALUE=256.0*FLOAT(DD)+INTMSK
02480.          IF(VALUE.GT.32767.0)VALUE=-1*(65536.-VALUE)
02490.C LOAD INTMSK AND DIRG WORDS     .-- 2 --
02500.          ARRAY(L+1)=INT(VALUE)
02510.C LOAD TARGET STEP COUNT         .---- 3 ----
02520.          ARRAY(L+2)=MAX
02530.          NFDF=491.0*MAX
02540.C CALCULATE AND LOAD RSD(6)
02550.          DO 110 K=1,6
02560.          IF(MN(K).NE.0)GO TO 95
02570.          MN(K)=Z'FFFF'
02580.95        VALUE=NFDF/FLOAT(MN(K))
02590.          IF(VALUE.GT.32767.)VALUE=-1.0*(65536.0-VALUE)
02600.          RSD(K)=INT(VALUE)
02610.100       IM=L+K+2
02620.C LOAD RELATIVE SPEED DIVISORS  .-- 4 TO 9 --
02630.110       ARRAY(IM)=RSD(K)
02640.C LOAD SPEED DIVISOR                    -- 10 --
02650.          ARRAY(L+9)=SDIV
02660.          BLOCK=BLOCK+1
02670.130       CONTINUE
02680.          RETURN
02690.          END
```

```
00010.;============== POSITION CONTROL PROGRAMS ==========
00020.;
00030.; CALLED FROM RUN AND ZERO TO CONTROL THE 'MCU'
00040.; ALTERS PRESENT ACTUATOR STATE VECTOR 'AS' (6 BY 1)
00050.;
00060.; CREATED  BY  GEOFF RATHBUN  9 / 7 / 85
00070.;
00080.;
00090.          ENTRY    SETUP,SCAN,DOS,GO
00100.          EXT      POSN
00110.; FORTRAN / ASSEMBLER ADDRESSES
00120.MACODE: DEFS     2
00130.NSTOP:  DEFS     1
00140.ZERO:   DEFS     1
00150.STPVEC: DEFS     2
00160.VALUE:  DEFS     2
00170.; ASSEMBLER / ASSEMBLER ADDRESSES
00180.DIRG:   DEFS     1
00190.LSTDRG: DEFS     1
00200.INTMSK: DEFS     1
00210.BLOCK:  DEFS     2
00220.TARGET: DEFS     2
00230.FSTREG: DEFS     1
00240.PORT:   DEFS     1
00250.RETURN: DEFS     1
00260.;
00270.;------- CONSTANTS ------------------------
00280.;
00290.; ROUTINE POSITIONS IN MEMORY
00300.MCU      EQU      0FCEEH
00310.CHAROT   EQU      033H
00320.; PORT CODES :
00330.STC1C    EQU      02H
00340.STC2C    EQU      06H
00350.STC3C    EQU      0AH
00360.STC1D    EQU      00H
00370.STC2D    EQU      04H
00380.STC3D    EQU      08H
00390.GPICC    EQU      012H
00400.GPICD    EQU      010H
00410.LATCH    EQU      14H
00420.CLKOFF   EQU      37E4H    ; MEMORY MAPPED FUNCTION
00430.CLKON    EQU      37E4H    ; MEMORY MAPPED FUNCTION
00440.; STC CODES
00450.MMPS     EQU      17H
00460.C1MPS    EQU      1H
00470.C2MPS    EQU      2H
00480.C3MPS    EQU      3H
00490.C4MPS    EQU      4H
00500.C5MPS    EQU      5H
00510.MM       EQU      400CH
00520.C1M      EQU      109H
00530.C2M      EQU      209H
00540.C3M      EQU      03E1H
```

```
00550.C4M      EQU      04E1H
00560.C5M      EQU      0BE1H
00570.LOAD     EQU      5FH
00580.LDARM    EQU      7FH
00590.LDAE5    EQU      6FH      ; LOAD & ARM 1-5 EXCEPT 5
00600.DISARM   EQU      0DFH
00610.STOP5    EQU      0D0H
00620.START5   EQU      30H
00630.SAVE12   EQU      0A3H
00640.; GPIC CODES
00650.AGPIC    EQU      0A1H
00660.DAGPIC   EQU      0A2H
00670.IMRSEL   EQU      0B0H
00680.CLRIRR   EQU      40H
00690.; PROGRAM CODES :
00700.ENTER    EQU      24D
00710.SPACE    EQU      16D
00720.BREAK    EQU      18D
00730.TRUE     EQU      0FFH
00740.FALSE    EQU      0H
00750.CR       EQU      0DH
00760.;============== SETUP ============ SETUP ==============
00770.;
00780.;-----------------------------------------------
00790.; 'STC' CHIPS 1 , 2 , 3  INITIALISATION ROUTINE
00800.;
00810.SETUP:   LD       B,3H
00820.C1:      LD       A,B
00830.         SLA      A
00840.         SLA      A
00850.         DEC      A
00860.         DEC      A
00870.         LD       E,A.; E = 4*B-2 (CONTROL PORT)
00880.         DEC      A
00890.         DEC      A
00900.         LD       D,A.; E = 4*B-4 (DATA PORT)
00910.; RESET AND LOAD STC'S 1,2,3
00920.         LD       C,E
00930.         LD       A,0FFH.; FFH IS RESET CODE
00940.         OUT      (C),A
00950.         LD       A,LOAD
00960.         OUT      (C),A
00970.; SET MASTER MODE REGISTER
00980.         LD       A,MMPS
00990.         OUT      (C),A
01000.         LD       HL,MM
01010.         LD       C,D
01020.         OUT      (C),L
01030.         OUT      (C),H
01040.; SET COUNTER 1 MODE REG
01050.         LD       A,C1MPS
01060.         LD       C,E
01070.         OUT      (C),A
01080.         LD       HL,C1M
```

```
01090.          LD      C,D
01100.          OUT     (C),L
01110.          OUT     (C),H
01120.; SET COUNTER 2 MODE REG
01130.          LD      A,C2MPS
01140.          LD      C,E
01150.          OUT     (C),A
01160.          LD      C,D
01170.          LD      HL,C2M
01180.          OUT     (C),L
01190.          OUT     (C),H
01200.; SET COUNTER 3 MODE REG
01210.          LD      C,E
01220.          LD      A,C3MPS
01230.          OUT     (C),A
01240.          LD      C,D
01250.          LD      HL,C3M
01260.          OUT     (C),L
01270.          OUT     (C),H
01280.; SET COUNTER 4 MODE REG
01290.          LD      C,E
01300.          LD      A,C4MPS
01310.          OUT     (C),A
01320.          LD      C,D
01330.          LD      HL,C4M
01340.          OUT     (C),L
01350.          OUT     (C),H
01360.; SET COUNTER 5 MODE REG
01370.          LD      C,E
01380.          LD      A,C5MPS
01390.          OUT     (C),A
01400.          LD      C,D
01410.          LD      HL,C5M
01420.          OUT     (C),L
01430.          OUT     (C),H
01440.; DECREMENT B WHICH CAUSES NEXT STC CHIP TO BE
01450.; LOADED ( 3 - 2 - 1 )
01460.          DJNZ    C1
01470.; LOAD AND ARM STC'S 2 & 3
01480.          LD      A,LDARM
01490.          LD      C,STC3C
01500.          OUT     (C),A
01510.          LD      C,STC2C
01520.          OUT     (C),A
01530.;-----------------------------------------
01540.; INTERRUPT STRUCTURE SETUP
01550.;
01560.          LD      HL,MCUISR
01570.          LD      (MCU),HL
01580.; RESET GPIC:
01590.          XOR     A
01600.          OUT     (GPICC),A
01610.; SET MODE BITS 4,3,2,1,0 TO '10010' BINARY GIVING
01620.; IREQ,GINT ACTIVE LOW,COMMON VECTOR,FIXED PRIORITY
```

```
01630.          LD      A,10010010B
01640.          OUT     (GPICC),A
01650.; PRESELECT, THEN SELECT AUTOCLEAR REGISTER TO
01660.; AUTOMATICALLY CLEAR ISR REG AFTER 'INTACK' SIGNAL
01670.          LD      A,11000000B
01680.          OUT     (GPICC),A
01690.          LD      A,0FFH
01700.          OUT     (GPICD),A
01710.; PRESELECT RESPONSE MEMORY OF LINE 0
01720.          LD      B,11100000B
01730.          LD      C,GPICC
01740.          OUT     (C),B
01750.; LOAD 'MCU' VECTOR AS COMMON VECTOR, LINES 0 - 7
01760.          LD      HL,MCU
01770.          LD      A,L
01780.          OUT     (GPICD),A
01790.; SETUP CPU FOR VECTORED INTERRUPTS
01800.          LD      HL,MCU
01810.          LD      A,H
01820.          LD      I,A
01830.; CLEAR GPIC OF ANY PENDING INTERRUPTS
01840.          LD      A,40H
01850.          OUT     (GPICC),A
01860.; DISABLE GPIC FROM ACCEPTING INTERRUPTS
01870.          LD      A,DAGPIC
01880.          OUT     (GPICC),A
01890.          RET
01900.;========= MCUISR =============== MCUISR =======
01910.;
01920.; CALLED BY 'MCU' AT EACH SYSTEM NODE. 'MCUISR'
01930.; CONTROLS TRANS-NODE OPERATIONS AND DATA MOVEMENT
01940.;
01950.; INPUTS: ADDRESS OF DIRECTION WORD
01960.;       : ADDRESS OF INTERRUPT MASK
01970.;       : ADDRESS OF NODE DELAY CODE
01980.;       : ADDRESS OF STOP CODE
01990.;
02000.MCUISR: EX      AF,AF'
02010.; STOP MOVEMENT BY TURNING OFF SPEED PULSES
02020.          LD      A,STOP5
02030.          OUT     (STC1C),A
02040.; SAVE COUNTERS 1 & 2 FOR SUBSEQUENT USE BY 'NODE'
02050.          LD      A,SAVE12
02060.          OUT     (STC1C),A
02070.          OUT     (STC2C),A
02080.          OUT     (STC3C),A
02090.; OUTPUT DIRECTION WORD AND GATE CONTROL BIT
02100.          LD      A,(DIRG)
02110.          OUT     (LATCH),A
02120.; LOAD ALL COUNTERS (FROM LOAD OR HOLD REGS, DEPENDING
02130.; ON LEVEL OF GATE CONTROL BIT JUST OUTPUT)
02140.          LD      A,LDARM
02150.          OUT     (STC3C),A
02160.          OUT     (STC2C),A
```

```
02170.; LOAD AND ARM STC 1 WHICH WILL RESTART SPEED GROUP
02180.          OUT     (STC1C),A
02190.; ARM SPEED DIVISOR GROUP ( C5/STC1 ) TO ENSURE IT
02200.; RESTARTS ( ATTEMPT TO ELIMINATE INTERMITTENT
02210.; FAILURE TO RESTART )
02220.          LD      A,START5
02230.          OUT     (STC1C),A
02240.          PUSH    HL
02250.; LOAD TARGET STEP VALUE INTO CHOSEN REGISTER
02260.          LD      A,(PORT)
02270.          LD      C,A
02280.          LD      A,(FSTREG)
02290.          OUT     (C),A
02300.          LD      HL,TARGET
02310.          DEC     C
02320.          DEC     C
02330.          OUTI
02340.          OUTI
02350.; PRESELECT 'IMR' FOR LOADING
02360.          LD      A,IMRSEL
02370.          OUT     (GPICC),A
02380.; OUTPUT INTERRUPT MASK TO 'GPIC'
02390.          LD      A,(INTMSK)
02400.          OUT     (GPICD),A
02410.; TABLE NOW GOING ON NEW DATA
02420.          EX      AF,AF'
02430.          PUSH    AF
02440.          PUSH    BC
02450.          PUSH    DE
02460.; UPDATE POSITION
02470.          CALL    NODE
02480.; RELOAD IN PREPARATION FOR NEXT INTERRUPT
02490.          CALL    RELOAD
02500.; TEST FOR A 'STOP AT NODE' COMMAND
02510.          LD      A,(NSTOP)
02520.          CP      TRUE
02530.          JR      NZ,CONT1
02540.; IF NSTOP = TRUE THEN PRINT THE UPDATED POSITION
02550.; AND STOP MOTION
02560.          LD      A,STOP5
02570.          OUT     (STC1C),A
02580.          LD      DE,NSTOPM
02590.          CALL    MESG
02600.          LD      HL,BLOCK
02610.          CALL    POSN
02620.; THEN SCAN FOR 'ENTER' OR 'BREAK'
02630.LOOP6:    LD      HL,3840H
02640.          BIT     0,(HL)
02650.          JR      Z,LOOP9
02660.; 'ENTER' HAS BEEN PRESSED  RESTART MOTION
02670.          LD      A,START5
02680.          OUT     (STC1C),A
02690.          JR      CONT1
02700.LOOP9:    BIT     2,(HL)
```

```
02710.          JR      NZ,BRK
02720.          JR      LOOP6
02730.; A BREAK HAS BEEN ORDERED,RETURN TO 'MENU'
02740.BRK:      LD      A,DAGPIC
02750.          OUT     (GPICC),A
02760.; SET RETURN FLAG TO TRUE
02770.          LD      A,TRUE
02780.          LD      (RETURN),A
02790.          POP     HL
02800.          POP     AF
02810.          POP     BC
02820.          POP     DE
02830.          EI
02840.          RETI
02850.; A PREMATURE INTERRUPT REQUEST MAY BE GENERATED
02860.; BEFORE THE CORRECT ALARM REG AND INTERRUPT MASK
02870.; IS SET. SO CLEAR 'IRR' TO ERASE THIS.
02880.CONT1:    LD      A,CLRIRR
02890.          OUT     (GPICC),A
02900.          POP     DE
02910.          POP     BC
02920.          POP     AF
02930.          POP     HL
02940.          EI
02950.          RETI
02960.; ========= LSTINT ========= LSTINT ==========
02970.;
02980.; THIS ROUTINE HANDLES THE LAST INTERRUPT IN A FILE
02990.;
03000.LSTINT:   PUSH    AF
03010.          PUSH    BC
03020.          PUSH    DE
03030.          PUSH    HL
03040.; STOP MOVEMENT
03050.          LD      A,DISARM
03060.          OUT     (STC1C),A
03070.; SAVE COUNTERS 1 & 2 FOR USE BY 'NODE'
03080.          LD      A,SAVE12
03090.          OUT     (STC1C),A
03100.          OUT     (STC2C),A
03110.          OUT     (STC3C),A
03120.; RESET VECTOR TABLE TO DIRECT INTERRUPTS TO 'INTER'
03130.          LD      HL,MCUISR
03140.          LD      (MCU),HL
03150.; DISABLE AND CLEAR GPIC
03160.          LD      A,CLRIRR
03170.          OUT     (GPICC),A
03180.          LD      A,DAGPIC
03190.          OUT     (GPICC),A
03200.; INFORM OPERATOR THAT MOVEMENT HAS ENDED.
03210.          LD      DE,ENDM
03220.          CALL    MESG
03230.; UPDATE AND PRINT POSITION
03240.          CALL    NODE
```

```
03250.          LD      HL,(BLOCK)
03260.          INC     HL
03270.          LD      (BLOCK),HL
03280.          LD      HL,BLOCK
03290.          CALL    POSN
03300.; SET RETURN FLAG TO 'TRUE'
03310.          LD      A,TRUE
03320.          LD      (RETURN),A
03330.          POP     HL
03340.          POP     DE
03350.          POP     BC
03360.          POP     AF
03370.          EI
03380.          RETI.; TO MAIN PROGRAM
03390.;=========== RELOAD ========= RELOAD ==========
03400.;
03410.; INPUTS: ADDRESS OF CURRENT MACODE BLOCK
03420.;       : ADDRESS OF CURRENT BLOCK NUMBER
03430.;
03440.;
03450.; PUT CURRENT MACODE ADDRESS INTO 'DE' REG
03460.RELOAD: LD      HL,(MACODE)
03470.          LD      E,(HL)
03480.          INC     HL
03490.          LD      D,(HL)
03500.          INC     HL
03510.; GET CURRENT BLOCK NUMBER
03520.          LD      BC,(BLOCK)
03530.; CHECK THAT FIRST MACODE VALUE=CURRENT BLOCK VALUE
03540.CONT6:  LD      A,D
03550.          CP      B
03560.          JR      NZ,ERROR
03570.          LD      A,E
03580.          CP      C
03590.          JR      Z,CONT2
03600.ERROR:  LD      A,OH
03610.          CP      C
03620.          JR      NZ,CONT4
03630.; WE'RE IN BLOCK 0, RETURN TO MAIN PROGRAM
03640.          LD      A,TRUE
03650.          LD      (RETURN),A
03660.          LD      DE,BLOCKO
03670.          CALL    MESG
03680.          LD      A,DISARM
03690.          OUT     (STC1C),A
03700.          LD      A,DAGPIC
03710.          OUT     (GPICC),A
03720.          RET
03730.; THE LAST BLOCK IS RUNNING.MODIFY THE VECTOR TABLE
03740.; TO DIRECT THE EXPECTED INTERRUPT TO 'LSTINT'
03750.CONT4:  LD      HL,LSTINT
03760.          LD      (MCU),HL
03770.; INCREMENT (BLOCK) FOR USE BY 'NODE'
03780.          INC     BC
```

```
03790.          LD      (BLOCK),BC
03800.          RET
03810.; NORMAL PATH CONTINUES HERE
03820.; INCREMENT AND SAVE BLOCK COUNTER INTO (BLOCK)
03830.CONT2:    INC     BC
03840.          LD      (BLOCK),BC
03850.; SAVE INTERRUPT MASK INTO (INTMSK) READY FOR 'INT'
03860.          LD      A,(HL)
03870.          INC     HL
03880.          LD      (INTMSK),A
03890.; DETERMINE (PORT) AND (FSTREG) FROM (INTMSK)
03900.          BIT     0,A
03910.          JR      NZ,B1
03920.          LD      A,2H
03930.          LD      (PORT),A
03940.          LD      A,7H
03950.          LD      (FSTREG),A
03960.          JR      B6
03970.B1:       BIT     1,A
03980.          JR      NZ,B2
03990.          LD      A,2H
04000.          LD      (PORT),A
04010.          LD      A,0FH
04020.          LD      (FSTREG),A
04030.          JR      B6
04040.B2:       BIT     2,A
04050.          JR      NZ,B3
04060.          LD      A,6H
04070.          LD      (PORT),A
04080.          LD      A,7H
04090.          LD      (FSTREG),A
04100.          JR      B6
04110.B3:       BIT     3,A
04120.          JR      NZ,B4
04130.          LD      A,6H
04140.          LD      (PORT),A
04150.          LD      A,0FH
04160.          LD      (FSTREG),A
04170.          JR      B6
04180.B4:       BIT     4,A
04190.          JR      NZ,B5
04200.          LD      A,0AH
04210.          LD      (PORT),A
04220.          LD      A,7H
04230.          LD      (FSTREG),A
04240.          JR      B6
04250.B5:       LD      A,0AH
04260.          LD      (PORT),A
04270.          LD      A,0FH
04280.          LD      (FSTREG),A
04290.; SAVE LAST 'DIRG' INTO (LSTDRG) FOR USE BY 'NODE'
04300.B6:       LD      A,(DIRG)
04310.          LD      (LSTDRG),A
04320.; SAVE NEW DIRECTION AND GATE BIT INTO (DIRG)
```

```
04330.          LD      A,(HL)
04340.          INC     HL
04350.          LD      (DIRG),A
04360.; SAVE TARGET STEP VALUE INTO (TARGET)
04370.          LD      E,(HL)
04380.          INC     HL
04390.          LD      D,(HL)
04400.          INC     HL
04410.          LD      (TARGET),DE
04420.; LOAD RELATIVE SPEED DIVISORS
04430.; BIT 7 OF (DIRG) DETERMINES DESTINATION OF DATA:
04440.; IF = 0, TO 'LOAD' REG,IF=1 TO 'HOLD' REG
04450.          LD      C,STC1D
04460.          LD      A,(DIRG)
04470.          BIT     7,A
04480.          JR      NZ,HOLD
04490.          LD      A,0BH
04500.          JR      OUT
04510.HOLD:     LD      A,13H
04520.; LOAD COUNTERS 3 & 4 WITH RELATIVE SPEED DIVISORS
04530.OUT:      OUT     (STC1C),A
04540.          LD      C,STC1D
04550.          OUTI
04560.          OUTI
04570.          INC     A
04580.          OUT     (STC1C),A
04590.          LD      C,STC1D
04600.          OUTI
04610.          OUTI
04620.          DEC     A
04630.          OUT     (STC2C),A
04640.          LD      C,STC2D
04650.          OUTI
04660.          OUTI
04670.          INC     A
04680.          OUT     (STC2C),A
04690.          LD      C,STC2D
04700.          OUTI
04710.          OUTI
04720.          DEC     A
04730.          OUT     (STC3C),A
04740.          LD      C,STC3D
04750.          OUTI
04760.          OUTI
04770.          INC     A
04780.          OUT     (STC3C),A
04790.          LD      C,STC3D
04800.          OUTI
04810.          OUTI
04820.; LOAD SPEED DIVISOR INTO 5/1
04830.          INC     A
04840.          OUT     (STC1C),A
04850.          LD      C,STC1D
04860.          OUTI
```

```
04870.          OUTI
04880.; SAVE CURRENT MACODE ADDRESS IN 'MACODE'
04890.          LD      (MACODE),HL
04900.          RET
04910.;========== GO ============= GO ==========
04920.;
04930.; INPUTS: ADDRESS OF FIRST MACODE VALUE
04940.;       : ADDRESS OF NSTOP CODE
04950.;       : ADDRESS OF STOP CODE
04960.;       : ADDRESS OF FIRST VALUE OF STEP VECTOR
04970.;
04980.; 'GO' PROGRAM IS ENTERED FROM THE FORTRAN PROGRAM BY
04990.; THE LINE:
05000.;         CALL GO(MACODE(1),NSTOP,ZERO,PP(1))
05010.; SAVE THE ADDRESSES OF THE ABOVE VARIABLES
05020.; SAVE ADDRESS OF MACODE(1):
05030.GO:       LD      (MACODE),HL
05040.; PUT NSTOP MESSAGE IN (NSTOP)
05050.          LD      A,(DE)
05060.          LD      (NSTOP),A
05070.          LD      H,OH
05080.          LD      A,(NSTOP)
05090.          LD      L,A
05100.          LD      (VALUE),HL
05110.; THE NEXT VARIABLE ADDRESSES ARE IN A DATA BLOCK
05120.; POINTED TO BY (BC)
05130.; LOAD ZERO CODE INTO (ZERO)
05140.          LD      A,(BC)
05150.          LD      L,A
05160.          INC     BC
05170.          LD      A,(BC)
05180.          LD      H,A
05190.          LD      A,(HL)
05200.          LD      (ZERO),A
05210.; LOAD ADDRESS OF PP(1) INTO (STPVEC)
05220.          INC     BC
05230.          LD      A,(BC)
05240.          LD      L,A
05250.          INC     BC
05260.          LD      A,(BC)
05270.          LD      H,A
05280.          LD      (STPVEC),HL
05290.; SET FIRST BLOCK NUMBER TO 0
05300.          LD      HL,OH
05310.          LD      (BLOCK),HL
05320.; LOAD FIRST SET OF DATA
05330.          CALL RELOAD
05340.; OUTPUT DIRECTION WORD AND GATE BIT TO LATCH
05350.          LD      A,(DIRG)
05360.          OUT     (LATCH),A
05370.; LOAD TARGET VALUE INTO APPROPRIATE REGISTER
05380.          LD      HL,TARGET
05390.          LD      A,(PORT)
05400.          LD      C,A
```

```
05410.          LD      A,(FSTREG)
05420.          OUT     (C),A
05430.          DEC     C
05440.          DEC     C
05450.          OUTI
05460.          OUTI
05470.; LOAD 'INTMSK' TO GPIC
05480.          LD      A,IMRSEL
05490.          OUT     (GPICC),A
05500.          LD      A,(INTMSK)
05510.          OUT     (GPICD),A
05520.; IS THIS ROUTINE BEING USED TO ZERO THE TABLE ?
05530.          LD      A,(ZERO)
05540.          CP      TRUE
05550.          JR      NZ,CONT9
05560.          LD      A,LDARM
05570.          OUT     (STC1C),A
05580.          OUT     (STC2C),A
05590.          OUT     (STC3C),A
05600.          LD      HL,(MACODE)
05610.          LD      DE,ZEROO
05620.          CALL    MESG
05630.          RET
05640.; NORMAL PROGRAM PATH (NON ZEROING) RESUMES HERE
05650.CONT9:    LD      HL,BLOCK
05660.          CALL    POSN
05670.; STOP 40 HZ SYSTEM CLOCK INTERRUPTS BY READING
05680.; LOCATION 37E4H. THIS DISABLES A FLIP/FLOP
05690.          LD      A,(CLKOFF)
05700.; RELOAD READY FOR INTERRUPT AT NODE
05710.          CALL RELOAD
05720.; ARM GPIC TO ACCEPT AND GENERATE INTERRUPTS
05730.          DI
05740.          LD      A,AGPIC
05750.          OUT     (GPICC),A
05760.; SEND 'GOING' MESSAGE OUT BEFORE ARMING STC'S
05770.; SO ANY EARLY INTERRUPT EXITS FROM 'GO' CODE
05780.          LD      DE,GOING
05790.          CALL    MESG
05800.; START MOTION
05810.          LD      A,LDARM
05820.          OUT     (STC3C),A
05830.          OUT     (STC2C),A
05840.          OUT     (STC1C),A
05850.; SETUP CPU TO VECTORED INTERRUPT MODE
05860.          IM      2
05870.; CLEAR IRR OF ANY PREMATURE INTERRUPTS
05880.          LD      A,CLRIRR
05890.          OUT     (GPICC),A
05900.          EI
05910.; THEN ENTER AND REMAIN IN 'SCAN' ROUTINE UNLESS
05920.; A KEYBOARD COMMAND DICTATES AN EXIT
05930.; 'MCUISR' INTERRUPTS THIS ROUTINE AT EACH NODE
```

```
05940.          LD      A,FALSE
05950.          LD      (RETURN),A
05960.LOOP7:    LD      A,(RETURN)
05970.          CP      TRUE
05980.          JR      NZ,B9
05990.; RESTART 40 HZ SYSTEM CLOCK
06000.          LD      (CLKON),A
06010.          IM      1
06020.          RET
06030.B9:       LD      HL,3840H
06040.          BIT     7,(HL)
06050.          JR      NZ,KEY2
06060.          JR      LOOP7
06070.KEY2:     LD      A,STOP5
06080.          OUT     (STC1C),A
06090.          LD      DE,STOP
06100.          CALL    MESG
06110.; EXIT OR RESTART MOTION ?
06120.LOOP8:    LD      HL,3840H
06130.          BIT     2,(HL)
06140.          JR      NZ,EXIT
06150.          LD      HL,3840H
06160.          BIT     0,(HL)
06170.          JR      Z,LOOP8
06180.; RESTART MOTION
06190.          LD      A,START5
06200.          OUT     (STC1C),A
06210.          LD      DE,GOING
06220.          CALL    MESG
06230.; CONTINUE SCANNING
06240.          JR      LOOP7
06250.EXIT:     LD      HL,ZERO
06260.          LD      A,0H
06270.          LD      (HL),A
06280.          LD      A,DAGPIC
06290.          OUT     (GPICC),A
06300.; RESTART 40 HZ SYSTEM CLOCK
06310.          LD      (CLKON),A
06320.          IM      1
06330.          EI
06340.          RET
06350.; ============ NODE ============= NODE ==========
06360.;
06370.; UPDATES THE PRESENT STEP POSITION VECTOR LOCATED
06380.; AT THE ADDRESS POINTED TO BY (STPVEC)
06390.; DATA COMES FROM HOLD REGS 1 & 2 OF STC'S 1,2,3
06400.NODE:     LD      IX,(STPVEC)
06410.; LOAD REG 'B' WITH LAST DIRECTION WORD
06420.          LD      A,(LSTDRG)
06430.          LD      B,A
06440.; LOAD REG 'DE' WITH CONTENTS OF H1/1
06450.; UPDATE STPVEC(1)  ( = PP(1) )
06460.          LD      A,11H
06470.          OUT     (STC1C),A
```

```
06480.          LD      C,STC1D
06490.          IN      E,(C)
06500.          IN      D,(C)
06510.          CALL    UPDATE
06520.; UPDATE STPVEC(2)
06530.; LOAD 'DE' WITH CONTENTS OF H2/1
06540.          LD      A,12H
06550.          OUT     (STC1C),A
06560.          IN      E,(C)
06570.          IN      D,(C)
06580.          CALL    UPDATE
06590.; UPDATE STPVEC(3)
06600.          LD      A,11H
06610.          OUT     (STC2C),A
06620.          LD      C,STC2D
06630.          IN      E,(C)
06640.          IN      D,(C)
06650.          CALL    UPDATE
06660.; UPDATE STPVEC(4)
06670.          LD      A,12H
06680.          OUT     (STC2C),A
06690.          IN      E,(C)
06700.          IN      D,(C)
06710.          CALL    UPDATE
06720.; UPDATE STPVEC(5)
06730.          LD      A,11H
06740.          OUT     (STC3C),A
06750.          LD      C,STC3D
06760.          IN      E,(C)
06770.          IN      D,(C)
06780.          CALL    UPDATE
06790.; UPDATE STPVEC(6)
06800.          LD      A,12H
06810.          OUT     (STC3C),A
06820.          IN      E,(C)
06830.          IN      D,(C)
06840.          CALL    UPDATE
06850.          RET
06860.; ========= UPDATE ======== UPDATE ============
06870.;
06880.; UPDATES STPVEC(N)
06890.; INPUT           - IX POINTS TO STPVEC(N)
06900.;                 - 'B' CONTAINS ROTATED 'DIRG'
06910.;                 - 'DE' CONTAINS STEPS MOVED BY LEG N
06920.; OUTPUT          - UPDATED (STPVEC(N))
06930.;
06940.; LOAD (STPVEC(N)) INTO 'HL'
06950.UPDATE: LD        L,(IX)
06960.          INC     IX
06970.          LD      H,(IX)
06980.; TEST APPROPRIATE BIT OF 'DIRG' ( IN 'B')
06990.          BIT     0,B
07000.; ADD OR SUBTRACT ACCORDING TO VALUE OF 'DIRG' BIT
07010.          JR      Z,SUBT
```

```
07020.; ADD MOVE TO OLD POSITION
07030.          ADD     HL,DE
07040.          JR      SAVE
07050.SUBT:     SCF
07060.          CCF     ;ENSURE CARRY FLAG DOESN'T AFFECT 'SBC'
07070.; SUBTRACT MOVE FROM OLD POSITION
07080.          SBC     HL,DE
07090.SAVE:     LD      (IX),H
07100.          DEC     IX
07110.          LD      (IX),L
07120.; INCREMENT IX TO POINT TO STPVEC(N+1)
07130.          INC     IX
07140.          INC     IX
07150.; ROTATE B 1 RIGHT READY FOR NEXT CALL
07160.          RRC     B
07170.          RET
07180.; ======== MESG ================ MESG ==========
07190.; OUTPUTS SEVERAL BYTES AS A MESSAGE USING 'CHAROT'
07200.;
07210.MESG:     LD      A,(DE)
07220.          PUSH    DE
07230.          CALL    CHAROT
07240.          POP     DE
07250.          INC     DE
07260.          CP      CR
07270.          RET     Z
07280.          JR      MESG
07290.;=========== DOS ============== DOS ========
07300.DOS:      IM      1
07310.          JP      402DH
07320.          RET
07330.; =========== SCAN ============= SCAN =========
07340.;
07350.; SCANS FOR AND RETURNS A KEYSTROKE CODE AS FOLLOWS
07360.; 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  G
07370.; 40 33 34 35 36 37 38 39 32 25 65 66 67 68 69 70 71
07380.;
07390.; H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X
07400.; 64 57 58 59 60 61 62 63 56 49 50 51 52 53 54 55 48
07410.;
07420.; Y  Z  ENTER  BREAK  SPACE
07430.; 41 42  24    18     16
07440.SCAN:     LD      DE,3880H
07450.          LD      B,08H
07460.LOOP1:    RLC     E
07470.          LD      A,(DE)
07480.          CP      00H
07490.          JR      NZ,LOOP2
07500.          DJNZ    LOOP1
07510.          LD      B,08H
07520.          JR      LOOP1
07530.LOOP2:    LD      C,B
07540.          LD      B,08H
07550.LOOP3:    BIT     0,A
```

```
   60.          JR       NZ,KEY
07570.          RLC      A
07580.          DJNZ     LOOP3
07590.          JR       SCAN
07600.KEY:      SLA      C
07610.          SLA      C
07620.          SLA      C
07630.          LD       A,C
07640.          ADD      A,B
07650.          LD       (HL),A
07660.          INC      HL
07670.          LD       B,OH
07680.          LD       (HL),B
07690.          DEC      HL
07700.          RET
07710.;-------- MESSAGES ----------------
07720.GOING:    DEFM      ' TABLE GOING PRESS ''SPACE'' TO STOP'
07730.          DEFB     CR
07740.STOP:     DEFM      ' TABLE STOPPED, PRESS ''ENTER'' TO GO'
07750.          DEFM      ' ''BREAK'' TO EXIT'
07760.          DEFB     CR
07770.BLOCKO:   DEFM      ' DATA FILE CORRUPT '
07780.          DEFB     CR
07790.NSTOPM:   DEFM      ' NODE: PRESS ''ENTER'' OR ''BREAK'''
07800.          DEFB     CR
07810.ZEROO:    DEFM      ' TABLE BEING ZEROED '
07820.          DEFB     CR
07830.ENDM:     DEFM      ' MOVEMENT COMPLETED'
07840.          DEFB      CR
07850.          END
```

Source file to cut a conical shape.

```
00010.          CO.,0.,570.,0.,15.,0.
00020.          V2.25
00030.          R1.0
00040.          PO.,0.,250.
00050.          CO.,0.,570.6,15.,15.,-15.
00060.          CO.,0.,571.2,30.,15.,-30.
00070.          CO.,0.,571.8,45.,15.,-45.
00080.          CO.,0.,572.4,60.,15.,-60.
00090.          CO.,0.,573.,75.,15.,-75.
00100.          CO.,0.,573.6,90.,15.,-90.
00110.          CO.,0.,574.2,105.,15.,-105.
00120.          CO.,0.,574.8,120.,15.,-120.
00130.          CO.,0.,575.4,135.,15.,-135.
00140.          CO.,0.,576.,150.,15.,-150.
00150.          CO.,0.,576.6,165.,15.,-165.
00160.          CO.,0.,577.2,180.,15.,-180.
00170.          CO.,0.,577.8,195.,15.,-195.
00180.          CO.,0.,578.4,215.,15.,-215.
00190.          CO.,0.,579.,225.,15.,-225.
00200.          CO.,0.,579.6,240.,15.,-240.
00210.          CO.,0.,580.2,255.,15.,-255.
00220.          CO.,0.,580.8,270.,15.,-270.
00230.          CO.,0.,581.4,285.,15.,-285.
00240.          CO.,0.,582.,300.,15.,-300.
00250.          CO.,0.,582.6,315.,15.,-315.
00260.          CO.,0.,583.2,330.,15.,-330.
00270.          CO.,0.,583.8,345.,15.,-345.
00280.          CO.,0.,584.4,360.,15.,-360.
00290.          CO.,0.,585.,15.,15.,-15.
00300.          CO.,0.,585.6,30.,15.,-30.
00310.          CO.,0.,586.2,45.,15.,-45.
00320.          CO.,0.,586.8,60.,15.,-60.
00330.          CO.,0.,587.4,75.,15.,-75.
00340.          CO.,0.,588.,90.,15.,-90.
00350.          CO.,0.,588.6,105.,15.,-105.
00360.          CO.,0.,589.2,120.,15.,-120.
00370.          CO.,0.,589.8,135.,15.,-135.
00380.          CO.,0.,590.4,150.,15.,-150.
00390.          CO.,0.,591.,165.,15.,-165.
00400.          CO.,0.,591.6,180.,15.,-180.
00410.          CO.,0.,592.2,195.,15.,-195.
00420.          CO.,0.,592.8,215.,15.,-215.
00430.          CO.,0.,593.4,225.,15.,-225.
00440.          CO.,0.,594.,240.,15.,-240.
00450.          CO.,0.,594.6,255.,15.,-255.
00460.          CO.,0.,595.2,270.,15.,-270.
00470.          CO.,0.,595.8,285.,15.,-285.
00480.          CO.,0.,596.4,300.,15.,-300.
00490.          CO.,0.,597.,315.,15.,-315.
00500.          CO.,0.,597.6,330.,15.,-330.
00510.          CO.,0.,598.2,345.,15.,-345.
00520.          CO.,0.,598.8,360.,15.,-360.
00530.          END
```

```
00010.C ============ MENU =============== MENU =========
00020.C
00030.C
00040.         LOGICAL MODE
00050.10       WRITE(5,1)
00060.1        FORMAT(1X,40('=')/
00070.      X  1X,' DATA GATHERING PROGRAMS'/
00080.      X  8X,'1 -- PRINT PRESENT STC 1 2 & 3 DATA'/
00090.      X  8X,'2 -- PRINT MACHINE CODE FILE DATA'/
00100.      X  1X,40('=')/)
00110.         READ(5,2)MODE
00120.2        FORMAT(I1)
00130.         IF(MODE.EQ.1)CALL STCDAT
00140.         IF(MODE.EQ.2)CALL MACDAT
00150.         GO TO 10
00160.         END
00170.C ============ MACDAT =============
00180.         SUBROUTINE MACDAT
00190.         INTEGER ARRAY(4000),BLOCK,FAST,PP(6),DIR,
00200.      X  DIRG,INTMSK,RSDF,MOVE,EOF,STOP,DELDIR,
00210.      X  V(6),INPUT
00220.         REAL M(6),SPEED,VE,RSDF,NFDF,VAL
00230.         REAL*8 H
00240.         LOGICAL A(8),B(8),C(8),FIRST
00250.         COMMON /LDCOM/H
00260.         EOF=Z'FFFF'
00270.         CALL LOAD(ARRAY,EOF)
00280.1        FORMAT(1X,'MACHINE CODE FILE: ',A7,'CONTAINS'//
00290.      X  1X,'BLOCK , INTMSK , DIRG  , FAST , STEP,
00300.      X   SPEED '/
00310.      X  20X,'MOVEMENTS'/20X,'POSITIONS'/
00320.      X  4X,'1',7X,'2',7X,'3',7X,'4',7X,'5',7X,'6'/)
00330.         FIRST=.TRUE.
00340.         BLOCK=0
00350.C GET STARTING POSITION
00360.         DO 20 I=1,6
00365.         INPUT=ARRAY(I)
00370.         CALL RDINT(INPUT,VAL)
00380.20       PP(I)=VAL
00390.         WRITE(2,3)(PP(I),I=1,6)
00400.3        FORMAT(1X,'STARTING POSITIONS:'/6(3X,I5))
00410.         WRITE(2,1)H
00420.C CALCULATE BLOCK LOCATION
00430.10       K=10*BLOCK+7
00440.         IF(ARRAY(K).NE.BLOCK)GO TO 60
00450.C GET 'INTMSK' AND 'DIRG' FROM BLOCK ELEMENT 2
00455.         INPUT=ARRAY(K+1)
00460.         CALL RDINT(INPUT,VAL)
00480.         VALUE=VAL/256.0
00490.         DIRG=INT(VALUE)
00500.         INTMSK=INT(256.0*(VALUE-DIRG))
00510.         CALL BINARY(INTMSK,A)
00520.         CALL BINARY(DIRG,B)
```

```
00530.C OBTAIN FASTEST MOTOR FROM 'INTMSK'
00540.         DO 30 I=1,6
00550.         J=9-I
00560.         IF(A(J).EQ.1)GO TO 30
00570.         FAST=I
00580.         GO TO 40
00590.30       CONTINUE
00600.C GET RSD OF FASTEST MOTOR
00610.40       L=K+FAST+2
00611.         INPUT=ARRAY(L)
00612.         CALL RDINT(INPUT,VAL)
00620.         RSDF=VAL
00630.C GET MOVEMENT OF FASTEST MOTOR
00632.         INPUT=ARRAY(K+2)
00635.         CALL RDINT(INPUT,VAL)
00640.         MOVE=VAL
00650.         NFDF=RSDF*FLOAT(MOVE)
00660.C CALCULATE SPEED OF FASTEST MOTOR
00662.         INPUT=ARRAY(K+9)
00665.         CALL RDINT(INPUT,VAL)
00670.         SPEED=4.9152E6/(RSDF*VAL)
00680.C CALCULATE MOVEMENT AND POSITION DATA
00690.         DO 50 I=1,6
00700.         I2=K+I+2
00702.         INPUT=ARRAY(I2)
00705.         CALL RDINT(INPUT,VAL)
00710.         M(I)=NFDF/VAL
00720.         V(I)=SPEED*M(I)/M(FAST)
00730.         DIR=-1
00740.         I3=9-I
00750.         IF(B(I3).EQ.1)DIR=1
00760.50       PP(I)=PP(I)+DIR*INT(M(I))
00770.         WRITE(2,4)BLOCK
00780.4        FORMAT(I5)
00790.C        WRITE(2,5)(A(I),I=1,6),(B(I),I=1,6)
00800.5        FORMAT(1X,2I1,2X,6I1,3X,8I1)
00810.         WRITE(2,6)(V(I),I=1,6),(PP(I),I=1,6)
00820.6        FORMAT(2(2X,6I6))
00830.         BLOCK=BLOCK+1
00840.         FIRST=.FALSE.
00850.         GO TO 10
00860.60       WRITE(2,7)
00870.7        FORMAT(1X,60('-')/)
00880.         RETURN
00890.         END
00900.C======== RDINT ================
00910.C READS AN INTEGER VALUE > 32767
00920.         SUBROUTINE RDINT(INPUT,OUTPUT)
00930.         INTEGER INPUT
00940.         REAL OUTPUT
00950.         IF(INPUT.LT.0)GO TO 10
00952.         OUTPUT=INPUT
00954.         RETURN
00956.10       OUTPUT=65536.+INPUT
```

```
00960.          RETURN
00970.          END
00980.C============= STCDAT ===========
00990.          SUBROUTINE STCDAT
01000.          REAL LD(3,5),HLD(3,5),M(3,5),A1(3),A2(3),
01010.         X  MM(3),COUNT(3,5)
01020.          LOGICAL LOAD,HOLD,MODE,C,D,H,L
01030.          WRITE(5,7)
01040.7         FORMAT(1X,'PRESS 2 FOR PRINTER,5 FOR VDU OUTPUT
01050.         X  '/)
01060.          READ(5,8)K
01070.8         FORMAT(I1)
01080.          DO 10 I=1,3
01090.          C=(I-1)*4+2
01100.          D=C-2
01110.          LOAD=8
01120.          HOLD=16
01130.          MODE=0
01140.          DO 20 J=1,5
01150.          LOAD=LOAD+1
01160.          CALL OUT(C,LOAD)
01170.          L=INP(D)
01180.          H=INP(D)
01190.          LD(I,J)=L+256*H
01200.          IF(LD(I,J).LT.0.0) LD(I,J)=65536.0+LD(I,J)
01210.          HOLD=HOLD+1
01220.          CALL OUT(C,HOLD)
01230.          L=INP(D)
01240.          H=INP(D)
01250.          HLD(I,J)=L+256*H
01260.          IF(HLD(I,J).LT.0.0) HLD(I,J)=65536.0+HLD(I,J)
01270.          MODE=MODE+1
01280.          CALL OUT(C,MODE)
01290.          L=INP(D)
01300.          H=INP(D)
01310.          M(I,J)=L+256*H
01320.          IF(M(I,J).LT.0.0) M(I,J)=65536.0+M(I,J)
01330.20        CONTINUE
01340.          CALL OUT(C,Z'BF')
01350.          HOLD=16
01360.          DO 30 J=1,5
01370.          HOLD=HOLD+1
01380.          CALL OUT(C,HOLD)
01390.          L=INP(D)
01400.          H=INP(D)
01410.          COUNT(I,J)=L+256*H
01420.          IF(COUNT(I,J).LT.0.0) COUNT(I,J)=65536.0
01430.         X  +COUNT(I,J)
01440.C RESTORE HOLD REG VALUES
01450.          R=HLD(I,J)
01460.          IF(R.GT.32768.0)R=-(65536.0-R)
01470.          H=INT(R/256.0)
01480.          L=256*(R-256*H)
01490.          CALL OUT(D,L)
```

```
01500.          CALL OUT(D,H)
01510.30        CONTINUE
01520.          CALL OUT(C,Z'07')
01530.          L=INP(D)
01540.          H=INP(D)
01550.          A1(I)=L+256*H
01560.          IF(A1(I).LT.0.0) A1(I)=65536.0+A1(I)
01570.          CALL OUT(C,Z'0F')
01580.          L=INP(D)
01590.          H=INP(D)
01600.          A2(I)=L+256*H
01610.          IF(A2(I).LT.0.0) A2(I)=65536.0+A2(I)
01620.          CALL OUT(C,Z'17')
01630.          L=INP(D)
01640.          H=INP(D)
01650.          MM(I)=L+256*H
01660.          IF(MM(I).LT.0.0) MM(I)=65536.0+MM(I)
01670.10        CONTINUE
01680.C OUTPUT DATA
01690.          WRITE(K,1)
01700.1         FORMAT(1X,'STC CHIPS 1,2 & 3 CONTAIN:'/
01710.      X   1X,'REGISTERS:',5X,'1',9X,'2',9X,'3',9X,'4',
01720.      X   9X,'5')
01730.          DO 40 I=1,3
01740.          WRITE(K,2)I,(COUNT(I,J),J=1,5)
01750.2         FORMAT(1X,'STC',I3/1X,'COUNTER :',5F10.0)
01760.          WRITE(K,3)(LD(I,J),J=1,5)
01770.3         FORMAT(1X,'LOAD    :',5F10.0)
01780.          WRITE(K,4)(HLD(I,J),J=1,5)
01790.4         FORMAT(1X,'HOLD    :',5F10.0)
01800.          WRITE(K,5)(M(I,J),J=1,5)
01810.5         FORMAT(1X,'MODE    :',5F10.0)
01820.          WRITE(K,6)A1(I),A2(I),MM(I)
01830.6         FORMAT(1X,' ALARM1 :',F8.0,' ALARM 2 :',F8.0,
01840.      X   ' MASTER MODE :',F8.0)
01850.40        CONTINUE
01860.          RETURN
01870.          END
01880.C ========= BINARY =============
01890.C CONVERTS AN 8 BIT NUMBER INTO 8 INTEGERS 1 OR 0
01900.          SUBROUTINE BINARY(INPUT,A)
01910.          LOGICAL A(8)
01920.          INTEGER V,INPUT,D(8)
01930.          DATA D/128,64,32,16,8,4,2,1/
01940.          DO 10 I=1,8
01950.          V=FLOAT(INPUT)/D(I)
01960.          A(I)=0
01970.          IF(V.LT.1)GO TO 10
01980.          A(I)=1
01990.          INPUT=INPUT-D(I)
02000.10        CONTINUE
02010.          RETURN
02020.          END
02030.C ======== LOAD ========= LOAD ===========
```

```
02040.          SUBROUTINE LOAD(ARRAY,EOF)
02050.          INTEGER ARRAY(4000),EOF
02060.          REAL*8 H
02070.          COMMON /LDCOM/H
02080.20        WRITE(5,1)
02090.1         FORMAT(1X,'WHAT IS FILENAME OF DATA TO BE',
02100.      X   ' LOADED?'/)
02110.          READ(5,2)H
02120.2         FORMAT(A6)
02130.          CALL OPEN(7,H,2)
02140.          DO 10 I=1,4000
02150.          READ(7,ERR=20,END=100)ARRAY(I)
02160.          IF(ARRAY(I).EQ.EOF)GO TO 100
02170.10        CONTINUE
02180.100       ENDFILE 7
02190.          RETURN
02200.          END
```